‘I liked it, but it made you think too much’: A case study of computer game authoring in the Key Stage 3 ICT curriculum.

Claire Johnson

Submitted for the qualification of Doctor of Philosophy

University of East Anglia
Faculty of Social Sciences
School of Education and Lifelong Learning

September 2014

Dedicated to my son Sam,
whose enjoyment of computer games
planted the seed for this study.

And to my mother and father,
who would have been quite chuffed
if they had known about it.

Abstract

The importance of giving pupils opportunities to become producers of digital media is well documented in the literature (see Harel, 1991; Papert, 1993; Kafai, 1995; Harel Caperton, 2010; Luckin et al., 2012; Nesta, 2012; Sefton-Green, 2013), however there has been little research in this area in the context of the UK Key Stage 3 ICT curriculum.

The purpose of this study is to achieve an understanding of how authoring computer games in a mainstream secondary setting can support the learning of basic game design and programming concepts. The research explores pupils' experiences of the process they followed and the areas of learning they encountered as they made their games, and considers what they valued and what they found difficult in the game authoring activity.

The research draws on the learning theory of constructionism, which asserts the importance of pupils using computers as 'building material' to create digital artefacts. In the process of creating these artefacts, over time, computers become 'objects to think with', enabling pupils to learn how to learn (Papert, 1980b; Harel and Papert, 1991a).

Data were collected in planning documents, journals and the games pupils made, in recordings of their working conversations, and in pair and group interviews. Findings indicate that as well as learning some basic programming concepts, pupils enjoyed the activity, demonstrated positive attitudes to learning and felt a sense of achievement in creating a complex artefact which had personal and cultural significance for them.

This research acknowledges the need to develop accessible units of work to implement aspects of the new Computing curriculum (DfE, 2013c), especially for teachers and pupils who have little prior knowledge of the field. It suggests that computer game authoring may offer a viable entry and considers the extent to which constructionist approaches are suitable for this kind of work.

Table of contents

List of tables

List of figures

Acknowledgements

I would like to express my heartfelt thanks to my supervisors, Professor Victoria Carrington and Professor Terry Haydn, for their invaluable guidance and encouragement, and to Dr John Woollard for setting me on the path and supporting me along the way.

Thanks also to the many online and offline colleagues who took the time to respond to queries and share their expertise, especially Dr Jake Habgood and Dr Simon Watts. I am indebted to Carly Sharples (Faculty Librarian) for helping me sort out my Endnote woes and for buying me all the expensive books I asked for!

I'm also grateful to Pad, Nick, and James for helping me with funding in the early days, to Jon for all the 'PhD conversations' and to Sam, Pad and Sarah for their unwavering moral support.

Special thanks are due to the pupils who took part in this study for their enthusiasm and hard work and for sharing their experiences and opinions so willingly.

And, of course, to my family for rooting for me and for keeping out of my way!

Claire Johnson 2014

Chapter 1 Introduction

1.1 Background

Debates surrounding the use of computers in education have a history that spans at least 30 years (Millwood, 2009). One argument which endures is that as part of their learning, young people should be given opportunities to use technology to create digital artefacts (Papert, 1980b, 1993; Luckin et al., 2012; Nesta, 2012; Sefton-Green, 2013). Yet there has been little research in this area in the context of the United Kingdom's (UK) Key Stage 3 ICT curriculum.

This study explores the introduction of a unit of work in computer game authoring delivered as part of the UK ICT[1] curriculum for Year 9 pupils. The research considers pupils' experiences of the activity, and sheds light on what they learned and what they found difficult when authoring computer games. Data were drawn from recordings of their working conversations, pair and group interviews, and by scrutinising the planning and design documents they produced and the computer games they created over an 8 week, 16 hour period.

The research was conducted at a time of flux in the UK secondary ICT curriculum, which, since the 2007 publication of the revised National Curriculum programme of study, has experienced turbulent changes, culminating in 2012 with the disapplication of the programme of study and the attainment target (Gove, 2012b). In 2013 the subject was redesignated as Computing and a programme of study for first teaching in September 2014 prescribes a new curriculum which incorporates at its core computer science, where pupils are taught "the principles of information and computation, how digital systems work, and how to put this knowledge to use through programming" (DfE, 2013c).

Against this background, computer game authoring emerges as an important area of inquiry, because it is an increasingly popular context in which to introduce programming concepts and practices at Key Stage 3.

[1] In this study, ICT (Information and Communication Technology) refers to the secondary curriculum subject taught in UK schools, as defined by the National Curriculum (DfES and QCA, 2004) and associated programmes of study (DCSF, 2008) in operation between 2007 and 2012.

While research interest has been shown in the area of computer game authoring, much of this research explores how authoring computer games is an important literacy, design or media activity (Robertson and Good, 2004; Buckingham and Burn, 2007b; Pelletier et al., 2010; Beavis et al., 2012; Merchant et al., 2013), and how it supports learning in a range of subjects in the primary phase (Harel, 1991; Kafai, 1995; Kafai and Resnick, 1996a; Baytak and Land, 2011b) or in out of school contexts (Peppler and Kafai, 2005; Kafai et al., 2009b). As Chapter 2 will show, little research relating to computer game authoring has been published from a UK secondary ICT perspective.

1.2 A brief history of the ICT curriculum

This section offers a brief history of the ICT[2] curriculum in UK secondary schools from 2000 onwards, which provides the background to and context of the current research.

Computers have been used in schools since the mid-1980s, but there was no IT curriculum at Key Stage 3 until 1988, when it was introduced as part of the National Curriculum for Design and Technology (Hammond et al., 2009). In 1995 Information Technology became a subject in its own right (DfE, 1995) and new requirements for teaching IT at Key Stage 3 were published (SCAA, 1995). When the National Curriculum was revised in 1999 IT was renamed ICT (Information and Communication Technology) (DfEE, 1999).

The 1999 National Curriculum set out what children should learn in Information Technology in five Strands of IT Capability (DfES, 1999):

- Communicating information (communicate in words, pictures and sounds).
- Handling information (gather, store and interrogate information).
- Modelling (explore models and simulations, write a control program, produce a spreadsheet).
- Measurement and control (use IT to control systems and equipment including sensors, monitor, measure and record data, develop sequences of instructions).
- Applications and effects (develop awareness of the role of new technology in the wider world).

[2] In considering the evolution of the subject (including its name), throughout this thesis, the term ICT is most often used, since that was the subject's appellation for the period in which the research was conducted.

At this time, ICT was a new subject in many schools in England and Wales (Hammond, 2004). Advisory schemes of work had been introduced in 2000 and consisted of fifteen sample teaching units (QCA, 2000). The ICT strand of the Key Stage 3 National Strategy (DfES, 2002a) introduced revised units between 2002 and 2003. Although not statutory, these became widely used as frameworks for what was taught and what software was used in schools (Lawson, 2010) and informed the content of many of the commercially produced textbooks, online schemes of work and curriculum guides published (for example, Leafline, 2003; Doyle, 2004; Furlonger and Haywood, 2004).

The emphasis in these materials was on the development of 'ICT capability' - and mainly involved the use of office productivity software and to a lesser extent, web design and control software, to enable pupils to develop a range of systems and publications and the knowledge, skills and understanding underpinning these. The contexts for the tasks were often related to the practices of businesses and organisations, and this was mirrored in coursework scenarios at Key Stage 4.

1.2.1 The Key Stage 3 sample teaching units

Sample teaching units 7.6, 8.5 and 9.1 (DfES, 2002b, 2003b, 2003c) delivered the programming content of the Framework in the context of the visual programming system *Flowol 2* (Bowker, 1998). Pupils learned that technology is used to control everyday events, such as the operation of traffic lights and car park barriers, automated greenhouses and theme park rides. They learned about the programming concepts of input, output, decisions, loops and sub-routines. Practical activities included developing and refining flowcharts to control simulations of such systems (see Figure 1). However, whilst these activities introduced pupils to the main building blocks of programming, they offered only closed systems with finite inputs and outputs.

Similar flowcharting exercises were included in the ill-fated Key Stage 3 ICT summative assessment test (QCA/RM, 2003), planned for first national rollout in 2008 but abandoned in 2007. 'Refocused' as optional 'formative assessment tasks' to support teacher assessment at the end of Key Stage 3, the test incorporated a task in 'sequencing instructions', which assumed knowledge of flowcharting software and required pupils to understand the use of variables, logical operators (<, >, =) and the terms 'input', 'process', 'output' and 'sub-routine' (QCA/NAA, 2008).

It becomes clear that the 'control and monitoring/sequencing instructions' element, as it appeared in the sample teaching units tasks and the test, was narrowly defined, and

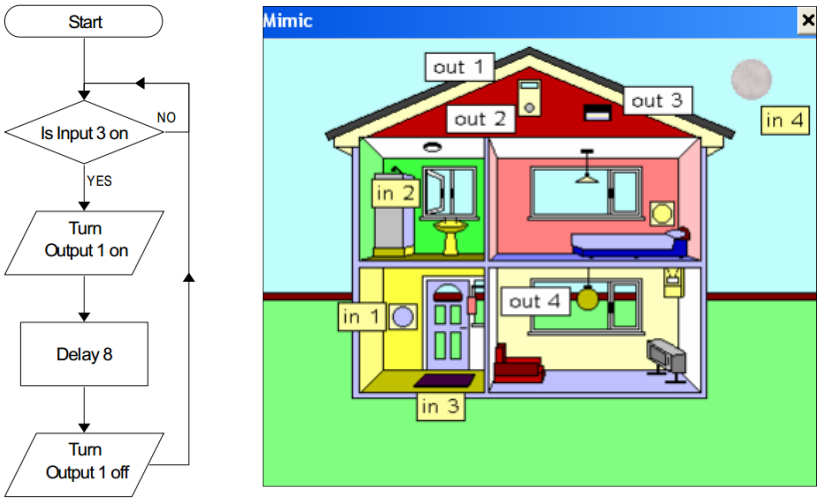


Figure 1: Flowol 2 burglar alarm flowchart

as a consequence, the learning, in terms of programming, afforded by these units (13 hours teaching time in total over 3 years), was somewhat limited.

1.3 Why computer game authoring?

Computer games are an integral part of many young people's lives. Recent figures suggest that eight in ten children aged 5-15 years play computer games using a fixed or portable games player at home (Ofcom, 2013). However, when it comes to *making* computer games, the UK figure is significantly lower, at 52% (Stokes, 2014).

1.3.1 The ICT curriculum

At the outset of the research in 2007, my interest lay in exploring the potential of computer game authoring as an engaging, relevant and challenging scenario for delivering the 'sequencing instructions' strand of the Programme of Study for Key Stage 3 (QCA, 2007b); according to OFSTED this area was the least well taught in the ICT curriculum (OFSTED, 2009; OFSTED, 2011). Since then, other imperatives have taken centre stage, as the following section describes.

From the mid-2000s there had been growing concerns in the ICT and computer science education community with the status and content of the subject. ICT was often taught by non-specialists and presenting information took precedence over processing and/or controlling information as the dominant output in many units of work (Peyton Jones et al., 2007; OFSTED, 2009; OFSTED, 2011). This concern with the content of

the secondary ICT curriculum and the uptake of computing at tertiary level was echoed internationally (see Tucker, 2006; Peyton Jones et al., 2007). Amongst stakeholders in higher education and industry there was a feeling that 'ICT' had usurped much of the content of latter-day 'computer studies' programmes (Webb and Cox, 2007). This disquiet heralded a call for the return of a 'computer science' element to the secondary ICT curriculum, spearheaded by the Computing at School group and supported by prominent industry players (see CAS, 2008a; CAS, 2008b; Peyton Jones, 2010; Livingstone and Hope, 2011; Schmidt, 2011; BCS, 2012; CAS, 2012a; Furber, 2012). One of the major examination boards, OCR, released a General Certificate of Secondary Education (GCSE) specification in Computing (OCR, 2010), which was piloted between 2010 and 2012 in response to this demand; new specifications for a GCSE in Computer Science were later announced in January 2012 by AQA, Edexcel and WJEC.

Amid this activity, the Secretary of State for Education delivered the death knell for ICT in his 2012 BETT speech (Gove, 2012b), in which he dubbed the subject 'offputting', 'demotivating' and 'dull' and announced the withdrawal of the programme of study from September 2012. A new programme of study for ICT was released in draft format in January 2013 (BCS, 2012), to take effect from September 2014, its content the subject of much debate. By February 2013 the National Curriculum Framework for consultation (DfE, 2013d) had recast the subject as 'Computing' to reflect substantially revised curriculum content and to remove the reported negative associations of ICT (DfE, 2013b); the programme of study was published in its final form in September 2013. This contained much more 'computing' content than previously, and the requirement to include at least one textual programming language at Key Stage 3 placed new demands on many ICT teachers (Nesta, 2014). Not surprisingly, the pedagogy of ICT has been described as unclear throughout this period (Webb, 2002; Hammond, 2004; Hadjerrouit, 2008).

Against this background, the importance of strengthening the delivery of programming becomes clear. The Programme of Study for Key Stage 3 ICT (QCA, 2007b), in operation from 2007-2012, referred to this area of learning as 'sequencing instructions', but until recently, very little 'traditional' programming using textual languages was learned at Key Stage 3. Visual programming software, such as *Flowol 2* (Bowker, 1998) had been commonly used to cover the control/programming element of ICT in many schools, largely because it was featured software in the National Strategy for ICT sample teaching units (DfES, 2002b, 2003b, 2003c). But the graphics were limited, the

scenarios not always compelling and the systems were not open-ended - an onscreen object animates along a prescribed path, a Belisha beacon flashes at a zebra crossing.

While the teaching of aspects of programming had always been present in the Key Stage 3 ICT curriculum, for GCSE ICT there had been no requirement to program at all, throughout the 2000s, although the topic appeared as an optional coursework unit (e.g. ICAA, 2001; OCR, 2009b) and as a question in examination papers (e.g. AQA, 2005; OCR, 2006), where pupils were required to sequence commands to draw a specified shape, for example. Whilst there were optional programming units in GCSE ICT and vocational Key Stage 4 specifications (e.g. OCR, 2009b; OCR, 2012a), it was not until 2010 onwards that GCSE specifications for Computing began to appear, which included a mandatory programming project (OCR, 2011; AQA, 2012a; Edexcel, 2012b; WJEC, 2012). At that point, the imperative of strengthening the learning of programming at Key Stage 3 began to gather momentum.

1.3.2 The return of game authoring

Whilst programming games had been popular in the 1980s in schools and homes, on computers such as Acorn's BBC model B, RM's 380Z and the Sinclair Spectrum, it became less common in the 1990s as the PC became the dominant system in school and the curriculum subject ICT emerged as a hybrid of Business Studies and IT (Hammond et al., 2009), focused on the use of 'office' software (Stevenson, 1997) and the development of systems to meet the needs of organisations.

By the late 2000s however, making computer games had resurfaced in ICT classrooms - promoted in the UK as part of the National Strategy for ICT (DCSF, 2008b) and in the Scottish Curriculum for Excellence outcomes (LTS, 2009). Following the National Curriculum revision in 2007 a new programme of study for Key Stage 3 ICT was released (QCA, 2007b), statutory from September 2008. A 'sequencing instructions' strand replaced the previous control and monitoring element, and this allowed schools a broader interpretation of the sort of learning in programming they could deliver and a wider choice in what type of software they could use to deliver this strand.

Subject leader development training materials to support the new framework (DCSF, 2008b) featured a 'sequencing instructions' task for Year 7 - to create a computer game for Year 3 pupils. This task offered an arguably more engaging and authentic learning experience than previous units; however, it was accompanied by other tasks (write a user guide, a test plan and an evaluation) - and was allocated 6 hours for

completion. The time given to programming the game itself was, at most, 3 hours and this had implications for what could be learned about the process.

The public release of *Scratch* (Resnick et al., 2003) in 2007, a media-rich tool designed to teach young people the basics of programming and computational thinking, was enthusiastically greeted by teachers and pupils alike, because it allowed much more creative freedom and agency than the control software referred to previously, had an easy entry point and was cost-free to schools. Similarly, free versions of game authoring software, such as *Game Maker 7* (YoYo Games, 2007), and commercial offerings such as *MissionMaker* (Immersive Education, 2007) brought the possibility of making games into the mainstream and introduced pupils to programming concepts and practices in the context of making a real, playable game - arguably a more rewarding outcome than the 'mimics' and simulations which had been the dominant fare of control software. As textbooks and other resources were released which included game authoring tasks (see Burtoft et al., 2008; Giles et al., 2008; Jones and Wilson, 2008; Reeves, 2008; Waller, 2009) the activity became increasingly common at Key Stage 3.

At Key Stage 4 specifications which included the creation of a computer game as an optional coursework task had been released earlier (e.g. Edexcel, 2006); DiDA's 'Games Authoring' unit was piloted in September 2009 (Edexcel, 2009). Since 2010, specifications for GCSE ICT, and vocational awards also offered game authoring units (OCR, 2009b; Edexcel, 2012c; OCR, 2012a), as do more recent GCSE specifications for Computing (AQA, 2012a; Edexcel, 2012b). Yet at the outset of the research in 2007, there was little prior learning of game authoring in the Key Stage 3 ICT curriculum to prepare pupils for these new Key Stage 4 courses.

There is no doubt that units of work featuring game design are being offered by more schools now than when the period of this research started (Repenning et al., 2010; Swacha et al., 2010). But little research which explores computer game authoring as part of the UK Key Stage 3 ICT curriculum has been undertaken.

The game authoring activity at the centre of this research had been originally designed to meet the learning objectives of the 'sequencing instructions' sub-strand of the National Curriculum for ICT (DCSF, 2008a). But this research also seeks to identify what other positive outcomes may occur when pupils author computer games, beyond the generic requirements of the National Curriculum Framework for ICT and the

programme of study, since much of the literature surrounding game authoring highlights factors beyond the learning of curriculum content alone as benefits of the activity (see Chapter 2).

1.3.3 Digital participation

Beyond preparing pupils for Key Stage 4 courses, there are other compelling arguments for including game authoring in the curriculum. Making digital media is important because young people need to be able to participate fully in the digital culture in which they live and make informed use of digital technology and media in their own lives (Hague and Williamson, 2009: 3). This involves becoming 'digitally literate', which foregrounds constructing/making/writing - not just reading/playing digital material (Papert, 1993; Salen, 2007; Resnick et al., 2009a; Harel Caperton, 2010). In particular, young people should be given the opportunity to be producers as well as consumers of computer games, since they are a significant cultural artefact (Robertson and Good, 2004; Habgood, 2006; Prensky, 2008; Williamson, 2009; Harel Caperton, 2010).

Others suggest that as a learning activity, computer game authoring gives value to pupils' prior, informal learning in playing computer games and bridges the gap between young people's use of technologies out of school, and the less wide-ranging uses of technology in schools (Buckingham et al., 2003). Introducing computer game authoring also develops young people's 'gaming literacy', enabling them to engage creatively *and* critically with this medium (Buckingham and Burn, 2007a; Salen, 2007). More recently, the importance of 'learning through making' with digital technologies has been reasserted (Nesta, 2012; Beckett, 2013; Mozilla, 2013a; Nesta et al., 2013; Sefton-Green, 2013), yet although a rising trend in practice is observed, it has not been subject to a great deal of research (Luckin et al., 2012).

1.3.4 Creativity

Creativity is identified in government education policy documents as one of seven dimensions which should permeate the curriculum (QCA, 2009). To foster creativity young people should be given opportunities to "appreciate the full range of … the creative industries" (QCA, 2009: 21). The new Computing programme of study states that pupils should "create … digital artefacts for a given audience" (DfE, 2013c: 2). But beyond government policy, there are other drivers for promoting game authoring as a creative practice in ICT education. Perhaps chief of these is that the ICT curriculum affords possibilities for particular sorts of creativity that are not so present elsewhere in the curriculum, since 'new technologies' possess distinctive features (identified as

'provisionality', 'interactivity', 'capacity', 'range', 'multi-modality' and 'social credibility') which can support creative practices that other media and tools might not offer (Loveless and Wegerif, 2004; Carbonaro et al., 2008).

Moreover, activities which may be termed 'creative', as opposed to practical or functional, should be part of the ICT curriculum in order to give pupils the opportunity to *be expressive with* technology (thus it includes game authoring, web page design, graphics, video, animation, audio production); these areas of learning were strongly supported in the research school. Beyond the 'surface' creativity that resides in the production of any computer generated outcome, there is an 'expressive' creativity that lies in being able to program, since programming offers the ability to create new uses for computers, rather than consuming the behaviours provided for us by others (Woollard, 2009).

Furthermore, activities such as game authoring, which involve learners in creating aspects of interactivity, are unique to the ICT curriculum (because they involve some sort of programming and because they involve the design of user interaction) and deserve greater understanding in terms of the nature of the learning involved.

Other rhetorics of creativity can also be invoked to argue the case for computer game authoring. Banaji and Burn (2007) refer to the idea of 'democratic creativity' and cultural re/production, which sees creativity as inherent in our cultural lives and resists the notion of creativity as the preserve of a minority of talented, gifted, artistic people - or residing only in certain cultural products (films, art, theatre, music). In this spirit, offering pupils opportunities to author computer games gives them access to a broader range of representational resources and enables them to engage with new sites of display (Jewitt, 2008).

1.3.5 Exploring ICT pedagogy

Introducing new curriculum content provides an opportunity to explore different approaches to learning. Making a computer game is a complex, extended activity (Harel, 1991; Kafai, 1995) and this has implications for pedagogy. The unit of work followed in the current study was an implementation of a constructionist learning activity, characterised by its collaborative work pattern, extended time frame and personally and culturally meaningful outcomes (see Chapter 3), and presents a scheme of work built around the use of *Game Maker 7* software (YoYo Games, 2007).

Pupils worked in pairs over an 8 week, 16 hour period to research, plan and make their games. In their pairs, they worked collaboratively, in so far as they pursued a single goal (Pritchard and Woollard, 2010: 62), negotiating and sharing their conceptions of the task and how to tackle its elements, co-constructing their understandings through interactions with each other and the software. At other times, they worked cooperatively within their working pairs, pursuing separate tasks (Stahl et al., 2006: 411), or with other members of the class, sharing their knowledge and showing others how to solve problems or achieve particular effects. Collectively, they worked as a 'community of practice' (Lave and Wenger, 1991) of novice game programmers, sharing resources, viewing each other's work in progress, interacting and learning together. The role of the teacher necessarily changed according to the different stages of the project. A more 'instructional' role was adopted at the start, as pupils learned the software and planned their games. Once underway with making their games, pupils took ownership of their learning in terms of how they managed their time and what tasks they completed. At this stage the teacher's role was focussed on guiding the process, providing resources and learning material, 'scaffolding' and troubleshooting, working with individuals and pairs rather than the class as a whole. The unit of work offered an extended, open-ended, multimodal experience, which contrasted with the more tightly-structured National Strategy sample teaching units delivered previously at the research school.

1.4 Selection of the game authoring software

One of the first tasks of the research was to establish which software would offer pupils the most accessible means to develop game authoring and basic programming skills, and to create a satisfying end-product in a relatively short period of time, an important consideration for this school-based research. *Game Maker* 7 was selected on the basis that its visual programming environment and drag and drop functionality offers an easy entry point, but a high ceiling in terms of enabling pupils of different abilities to learn about game authoring and programming concepts and practices.

A second reason for selecting *Game Maker* is that it is widely used in UK schools and supported by several textbooks and resources aimed at Key Stage 3 (see Giles et al., 2008; Jones and Wilson, 2008; Reeves, 2008; Waller, 2009). Current GCSE ICT and Computing specifications feature *Game Maker* in their sample assessment material (OCR, 2012b), as recommended software (AQA, 2012d; Edexcel, 2012a; OCR, 2012b)

and in their training events (see AmmA, 2011; AQA, 2012d). The software is also increasingly used in education internationally (from primary to tertiary levels) to deliver elements of learning in programming (see Claypool and Claypool, 2005; Baytak et al., 2008; Whitehead, 2008; Dalal et al., 2009; Gamble, 2009; Hernandez et al., 2010; Hoganson, 2010; Kuruvada et al., 2010b).

However, it should be noted that *Game Maker* was not originally written for educational use. Rather, it was developed to promote children's interest in game design and programming for recreational/hobbyist purposes, whilst also incorporating features to appeal to older users with some programming knowledge (Overmars, 2015). Publicly released in 1999, as a free download, *Game Maker* first attracted an amateur audience, only later making its entry into educational settings. At tertiary level it has been used as a rapid prototyping tool to teach game design principles to students who already knew how to program (Habgood, 2013). But the idea that the software should enable novices to learn to program with visual symbols first and then progress to using its built-in textual language (GML) was an important design principle from the start.

Yet, despite its widespread use, there is little published research on whether making games with such environments leads to increased understanding of programming concepts (Denner et al., 2012) and at the time of writing, no research has been published in how *Game Maker* is used to support the teaching and learning of programming in the UK Key Stage 3 ICT curriculum.

1.5 Authoring games with Game Maker

Game Maker enables users to create two-dimensional (2D) computer games without any prior programming experience. Instead of writing textual code, users drag and drop graphical representations of functions and other programming components to define game objects' behaviours. Figure 2 below illustrates the graphical programming icons on the right.

Game Maker also includes a 'conventional' textual programming language, GML, which allows users to move on from visual programming as their understanding of programming constructs develops, and may be conceived as an 'ideal tool' for learning to program since its functionality can be extended alongside the user's growing

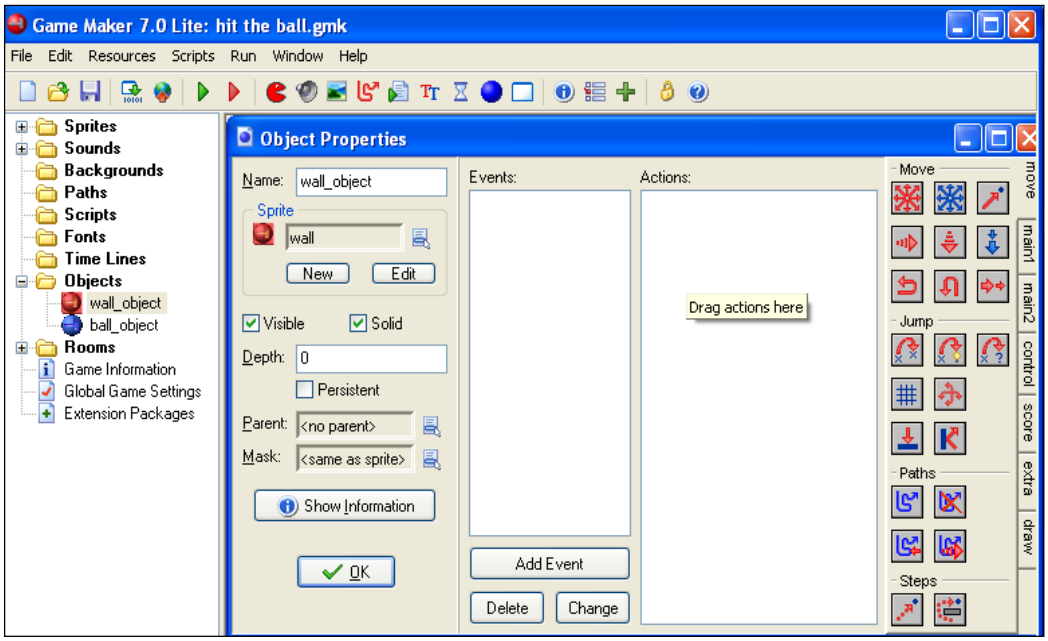


Figure 2: The Game Maker 7 interface

capability (Yatim and Masuch, 2007). This scalability of use was an important factor in the selection of *Game Maker* for this study.

1.5.1 The structure of a Game Maker game

A *Game Maker* game is made up the following components:

Rooms - the game space - each level of a game will have its own room.
Backgrounds - graphical resources which are loaded into the room to create the appearance of the game world.
Objects - programmable entities which exhibit behaviours and possess attributes. Player and non-player characters interact with objects according to the events and actions specified for them.
Sprites - the graphical representations of an object.
Events - inputs assigned to objects trigger their behaviour - a keyboard press, a mouse click, a collision of objects on screen, for example.
Actions - actions assigned to an event produce an output - an object may bounce, a sound may play, points may be scored, for example.

Figure 3 shows how these components are structured.

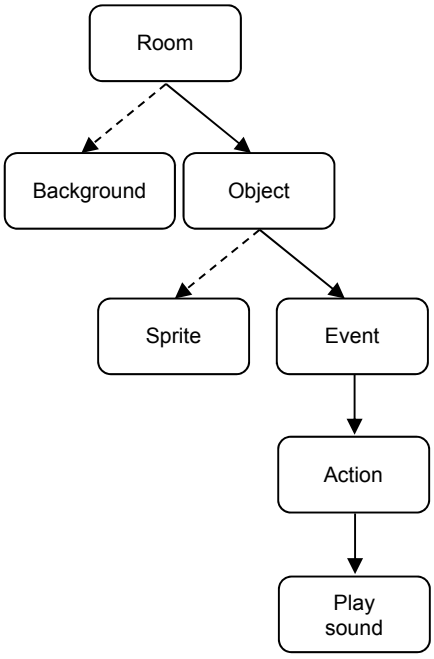


Figure 3: The structure of a game in Game Maker (Kirk, 2006: 31)

Pupils learn how to combine these components and how to select and sequence events and actions to create a playable game. For each action assigned to an object, pupils have to make choices which define the game play itself. For example they must specify the speed and direction of objects, what happens if the player loses all their lives, when a sound should play, how the score should increase. In so doing they have to use mathematical concepts (negative number, the use of coordinates, relative and absolute value), physical concepts (position, speed, acceleration, collisions), and programming concepts (sequence, conditions, variables, loops).

1.6 The Research

The research was conducted with one group of Year 9 pupils (13-14 year olds), over an 8 week period (16 hours) at a mixed, comprehensive school in South East England. The researcher was the group's timetabled teacher and Curriculum Leader for ICT.

The research explores the implementation of a unit of work in computer game authoring as part of the Key Stage 3 ICT curriculum, in which pupils constructed a game narrative, created or sourced game assets (graphics, background music, sound effects), and designed and programmed game object and player interactions and game play.

Game Maker 7, the software used in this research, does not require sophisticated programming; events and actions are selected from libraries of pre-programmed 'blocks' and compiled to create the game and its interactivity. This research seeks to explore the extent to which authoring computer games using *Game Maker* can support the learning of basic game design and programming concepts. The emphasis is on what learning, specific to the ICT curriculum, takes place when pupils author computer games, rather than on the literacy, narrative development, or media aspects of game authoring, which is the area of interest of much of the research literature (see Good and Robertson, 2006b; Robertson and Good, 2006; Buckingham and Burn, 2007b; Robertson and Nicholson, 2007; Hayes and Games, 2008; Harel Caperton, 2010; Beavis et al., 2012; Merchant et al., 2013).

In this it makes an important contribution to the field of computer science education, since little research has been targeted at secondary level (Begel and Klopfer, 2004), and there have been few studies of whether authoring computer games increases children's understanding of computer science concepts (Denner et al., 2012), or what kind of knowledge students learn from creating games using visual programming languages (Koh et al., 2010). Moreover, there are few studies which look at the learning of computing concepts through game authoring *within a classroom setting* (Wilson et al., 2012).

Both the activity and the research draw on the learning theory of constructionism, described in more detail in Chapter 3. Constructivism asserts that learners construct their own knowledge and understanding, based on their prior experiences and that learning is socially and situationally mediated (Pritchard and Woollard, 2010). Constructionism extends this idea by suggesting that learning is more successful when it arises out of learners constructing a real artefact (Papert, 1991b).

1.6.1 Research questions

The study seeks to address the following research questions:

- What are pupils' perceptions about the process and outcomes of their learning during a constructionist-designed game authoring activity?
- How does computer game authoring using *Game Maker* support the learning of basic programming concepts and practices?
- What difficulties do pupils have with game authoring (game design and game programming)?

- What affective value is there to pupils in authoring computer games?

In exploring these questions, a greater understanding of the pedagogy of computer game authoring in the ICT curriculum will be achieved. The need for the ICT curriculum to place greater emphasis on computing has been outlined above. The context for introducing game authoring as one way of doing this has been set alongside the use of flowcharting software, which was widely adopted for learning programming at Key Stage 3 previously (see DfES, 2002b, 2003b, 2003c). The need to provide an accessible introduction to basic programming concepts for teachers and pupils who do not have a strong background in computer science is important in the current context (see Nesta, 2014).

1.6.2 Assumptions

This study examines the proposition that there is value to introducing computer game authoring in the Key Stage 3 ICT curriculum and seeks to explore where specifically that value lies. It is suggested that computer game authoring provides an accessible and motivating introduction to the practices and concepts of basic computer programming. This assumption is made on the basis of the researcher's experience with similar activities in an extra-curricular context, where noticeable positive motivational effects were observed. The assumption is also supported by the literature surveyed in Chapter 2, relating to the practice of computer game authoring in schools and after school programmes. It is further supported by the emergence of examination specifications in the UK which offer electives in computer game authoring activities. In academia, a growing number of UK conferences feature game authoring and/or game-based learning in their programmes (e.g. DIGRA 2014, Games Britannia 2014, European Conference on Games Based Learning 2014, Games Learning Society 2014, Serious Play 2014); publications in academic journals relating to computer game authoring also suggest that there is value to the activity (see Chapter 2).

1.7 Structure of the thesis

This chapter has set the scene for the research. In Chapter 2 I survey the literature surrounding game authoring and follow on from this to outline the theoretical framework for the research in Chapter 3. Chapter 4 describes the research design and methods used to capture and analyse the data. Chapters 5-9 consider the research findings and Chapter 10 presents a discussion and conclusion.

Chapter 2 A literature review

2.1 Introduction

In this chapter I review the literature relating to computer game authoring thematically, grouping research projects by their primary focus: motivation; literacy and narrative development; digital, game and media literacy; programming. It becomes clear that there is a gap in the research literature in studies of game authoring in the UK secondary ICT curriculum. This is not surprising, given that the activity has only recently entered the mainstream, as games technologies, the rise of digital media, software and curricular developments have converged to make it a possibility.

2.1.1 The scope of the literature review

The literature review confines its scope to research relating to the use of software to *create* computer games in mainstream and informal educational settings. It refers only in passing to the field of game-based learning (the use of commercial computer games and game technologies for education), since the focus of the present research is on making games, rather than playing games, for learning.

A significant amount of research has been conducted in the field of game-based learning, as evidenced in recent overviews (see Pivec, 2009; Perotta et al., 2013), but until recently, less interest has been shown in computer game authoring (Baytak and Land, 2010). In much of this research, the focus has been on how game authoring supports the learning of mathematics at primary level (Harel, 1991; Kafai, 1995; Shaw et al., 2012; Ke, 2014), narrative or literacy development (Robertson and Good, 2004, 2005; Carbonaro et al., 2005; Howland et al., 2013) or how it adds to the secondary Media Studies curriculum (Buckingham and Burn, 2007b; Pelletier et al., 2010). Other research focuses on the motivational benefits of creating games and on the social learning that surrounds it (Robertson and Howells, 2008; Jung and Park, 2009; Molins-Ruano et al., 2014), but there is little research which explores the learning benefits of creating computer games from the UK Key Stage 3 ICT perspective, and which takes account of the realities of school (Perotta et al., 2013).

2.2 Game-based learning

Although interest in the potential of computer games for education can be traced back to the 1970s and 80s (O'Neil et al., 2005; Bragge and Storgards, 2007), the rise in popularity of computer games has led to a surge in research in the area of game-based learning, as indicated in continuing publication of overviews of the field over the past 13 years (see Prensky, 2001; Gee, 2003b; Kirriemuir and McFarlane, 2004; Mitchell and Savill-Smith, 2004; Sandford and Williamson, 2005; Squire, 2005; Becta, 2006; de Freitas, 2006; Egenfeldt-Nielsen, 2006; ELSPA, 2006; Sandford et al., 2006; Bryant et al., 2007; Pivec and Pivec, 2008; Ke, 2009; Klopfer et al., 2009; Pivec, 2009; Williamson, 2009; LTS, 2010; Bray, 2011; Felicia, 2011; Tobias and Fletcher, 2011; Connolly et al., 2012; Hwang and Wu, 2012; Larsen McClarty et al., 2012; Felicia, 2013; Perotta et al., 2013). It is beyond the scope of this study to review this literature but a brief summary of some points of interest relating to the use of games in schools is presented here, as a background to, and to distinguish this research area from the current study, which is concerned with exploring computer game authoring as part of the Key Stage 3 ICT curriculum.

Becta's 'Computer Games in Education Project' (Becta, 2006), identifies the benefits of learning with games to be increased motivation and collaboration and development in ICT and thinking skills. In its second 'Emerging Technologies for Learning' report (Bryant et al., 2007) Facer et al. suggest that rather than simply using commercial computer games in the classroom, more attention should be paid to how principles of games-based learning "might inform the creation of radically new learning environments" (Facer et al., 2007: 52), echoing Prensky's claim that "as a learning tool, computer games may be the most powerful mechanism ever known" (Prensky, 2002: 2).

Yet in the mid-2000s, the empirical evidence for the effectiveness of games as learning environments was scant (O'Neil et al., 2005). While some research claimed that learning with games was only effective when supported by effective instructional measures (Egenfeldt-Nielsen, 2006), other reports recorded a negative effect of games on learning, but a positive effect on motivation (Rieber, 2005). More recent reviews find that the most frequently occurring impacts are improved knowledge acquisition and understanding and affective and motivational outcomes (Connolly et al., 2012: 661). Other publications encourage and validate the use of computer games for learning and suggest the need for a pedagogy of games (Ulicsak and Williamson, 2010). NFER's

latest survey of games based-learning in schools (Perotta et al., 2013) suggests that there is a split in the literature regarding the extent to which computer games impact on academic achievement; some studies observe improvements and others do not. Attitudes to learning are improved when games and simulations are used, compared to traditional methods, but the evidence for improved attainment is less secure (ibid.). However, it is a consistent finding that computer games have a positive impact on problem-solving skills, broader knowledge acquisition, motivation and engagement.

As the interest in game-based learning in schools increases, the development of games to support learning and teaching across the curriculum continues to grow (Larsen McClarty et al., 2012). Where once the use of commercial off-the-shelf games was considered to be an innovative approach to increase engagement and attainment in subjects such as literacy (Sandford and Williamson, 2005; Rylands, 2007) and history (Squire, 2004) for example, in subsequent years the 'gamification' of learning material and the use of games as a teaching strategy has become commonplace (see for example, Bryant et al., 2007; Robertson, 2009; LTS, 2010). Nowadays commercially produced teaching resources and online educational programmes which include games designed to teach subject specific material proliferate (Bober, 2010). In the United States (US) the Quest to Learn school is designed with game-based learning as its defining feature (Salen et al., 2011). The notion of gamification has also spilled over into assessment practices, typified by the digital badge movement (see Mozilla, 2013b) and the use of points and reward features in online learning environments such as Khan Academy (Khan Academy, 2012).

In short, although there is much theoretical support for the benefits of digital games in learning and education, there is mixed empirical support (Larsen McClarty et al., 2012; Perotta et al., 2013), yet it is clear that as a pedagogical tool, game-based learning is gaining a stronghold. This may be because it offers an alternative learning experience to traditional approaches and because such playful, immersive environments require (inter)active participation, which alongside their dynamic, multimodal qualities and immediate feedback provide motivating learning encounters.

2.2.1 Game-based learning in ICT

In terms of the curriculum subject ICT, there is a lack of empirical evidence regarding the learning of computer science concepts via game-based approaches within school settings (Papastergiou, 2009), yet studies which do exist suggest that using a gaming approach is more effective in promoting students' knowledge of computing concepts,

and more motivational than a non-gaming approach (ibid.) and that students experience reduced task difficulty and anxiety levels, and increased motivation when using games to learn programming concepts, compared to learning experiences in traditional lectures (Liu et al., 2011).

Although several games have been developed to support the teaching and learning of computing topics, including programming, these are developed as academic research projects at tertiary level (Harteveld et al., 2014) (see for example, Natvig and Line, 2004; Barnes et al., 2008; Chaffin et al., 2009; Yeh, 2009; Muratet et al., 2011; Schmitz et al., 2011; Kazimoglu et al., 2012) and to my knowledge are not widely used in schools. Other games such as *Robocode* (Nelson, 2001), *Lightbot* (Yaroslavski, 2008), *Robozzle* (Ostrovsky, 2009), *CodeCombat* (Saines et al., 2013) and *Hakitzu* (Kuato, 2013) have some potential for the secondary sector ICT/Computing curriculum, but have not been the subject of recent academic research.

Research projects which focus on the development of game-based learning environments for teaching computer science at secondary level are exemplified by *Engage* (Rodríguez et al., 2013), a three-dimensional (3D) platformer game in which pupils solve puzzles in the form of programming tasks, and *CodeSpells* (Esper et al., 2013), a game which immerses programming into game play to teach middle school students to code in Java. This work seeks to identify successful programming practices including: collaboration and the role of dialogue in supporting pupil engagement (Rodriguez et al., 2013); learner-structured, self-driven activities; access to immediate feedback and support; the importance of exploration, creativity, and play, and the creation of meaningful artifacts (Esper et al., 2013). Another study describes *Gram's House*, a game designed to teach computing concepts and to appeal to girls' interests, developed as part of an ongoing research project to encourage girls to study computing in high school and beyond (Stewart-Gardiner et al., 2013). Yet despite the promise of this research, these learning environments have not yet migrated to mainstream school settings and the extent to which such research informs or impacts on the current UK situation is unclear, although discussions about the pedagogy of programming have begun to surface in forums and publications concerned with the teaching of computer science (e.g. CAS, 2008a; CAS, 2014).

2.3 Research in computer game authoring

Notwithstanding the importance of the research into game-based learning, the present study is concerned with pupils' experience of making their own computer games, and the following section surveys the literature in this increasingly mainstream practice.

Computer games are a significant cultural artefact (Fromme and Unger, 2012) and the idea that pupils should be given the opportunity to be producers as well as consumers of computer games is strongly supported in the literature (see Kafai, 1995; Robertson and Good, 2004; Habgood, 2006; Buckingham and Burn, 2007a; Salen, 2007; Hague and Williamson, 2009; Williamson, 2009; Harel Caperton, 2010; Li, 2010).

Yet although the idea of game authoring as an educational activity is gaining in popularity, evidence of the learning outcomes of game authoring is not well documented (Tiong and Yong, 2008). There are few published studies to date on the effects of computer game development as a pedagogical activity generally (Owston et al., 2009) or in teaching computing in particular (Smith and Grant, 2000). There is also little evidence of the role of game design and programming in digital literacy development (Harel Caperton, 2010); more broadly, the educational potential of construction activities with digital media is not well explored (Zorn, 2009). Few formal studies involving children and adolescents have been conducted (Carbonaro et al., 2008; Baytak and Land, 2010) and there has been little research into the specific field of game development education (Northcott and Miliszewska, 2008). In particular, there is little published research on how *Game Maker*, the authoring tool used in this research, has been used by educators to teach programming concepts at secondary level (Hayes and Games, 2008; Daly, 2009).

Despite a growing number of accessible tools for digital media creation the educational benefits of these programming languages are rarely the focus of research (Stolee and Fristoe, 2011) and although the importance of 'learning through making' with digital technologies has been reasserted by public bodies and industry more recently (Nesta, 2012; Beckett, 2013; Mozilla, 2013c; Sefton-Green, 2013; Aardman/Nominet Trust, 2014), it is acknowledged that the observed rising trend in practice has not been adequately researched (Luckin et al., 2012). In particular, few studies are focused on game making within classrooms (Wilson et al., 2012) and within the context of ICT education in the secondary phase. The present study is designed to contribute to the knowledge base of this, to date, poorly documented but increasingly important area.

2.4 Game authoring and motivation

It is a consensus in the literature that young people find computer game authoring motivating (see Harel, 1991; Kafai, 1995; Howland et al., 1997; Chamillard, 2006; Kafai, 2006b; Denner, 2007; Sanford and Madill, 2007b; Repenning and Ioannidou, 2008; Cheng, 2009; Jung and Park, 2009; Fowler and Cusack, 2011; Robertson, 2012, 2013; Hwang et al., 2014; Ke, 2014; Molins-Ruano et al., 2014).

Key publications in this area focus on game design as a source of motivation in the primary phase. The 'Adventure Author' project (Robertson and Howells, 2008) studied a Year 5 class who made computer games over an 8 week period. The analysis notes the enthusiasm and motivation for learning and the determination to reach high standards which arose out of the activity. Making games is motivating, the authors argue, because learners are actively engaged in authentic, rich tasks and exercise a variety of skills (creating characters, dialogue, and the visual design, and programming the action) to create a complex artefact. In making games, young people can pursue their own lines of enquiry, and in so doing develop a sense of ownership and self-determination, which are powerful levers for learning. Motivation also arises because the resulting artefact is of value in popular culture and can be enjoyed by a real audience (Robertson and Howells, 2008). Moreover, the use of game authoring software itself is motivating, since pupils can learn autonomously using the software as a sounding board for their ideas.

Other studies find that making games is motivating because it offers a playful way of learning (Li, 2010). Li's study of 21 primary, summer camp pupils reports that they were highly engaged when making games and found the activity 'fun', which led to increased commitment for learning. In this mixed methods case study, most students displayed positive emotions whilst making their games and felt proud of their completed versions. They valued the autonomy they were given and expressed a sense of ownership and control of their learning. Also motivating was the new identity they adopted as 'teacher', showing peers how to use the software and how to improve their games. Motivation and engagement were also evidenced in their persistence in problem-solving. Errors and problems were frequent occurrences but such challenges spurred pupils on, because they were real problems to be solved and directly linked to their final game outcomes.

Similar findings are reported in a study of secondary pupils who made games to learn about computer programming (Sanford and Madill, 2007a). Making games motivated these pupils because in creating something important to themselves they experienced feelings of empowerment and ownership (ibid.: 585). Giving and using peer feedback, and playing each other's games were rewarding for pupils and they felt great satisfaction and a sense of achievement in creating a challenging product. Pupils were also motivated by the feedback provided by the software, which allowed them to see their own progress through the running of the game.

Game authoring is also motivating because it gives pupils the freedom to create original characters and gameplay. When making games, pupils draw upon and reflect their own interests and preferences in the game genre, characters, setting, and interfaces they choose to create and gain a perception of themselves as producers and originators. This ability to create and manipulate one's own products is a unique motivational catalyst (Howland et al., 1997).

In an Australian study conducted with 20 schools (DEECD, 2010), findings report that pupils were motivated by and engaged when making games with *Kodu* (Microsoft Research, 2009) because they enjoyed being able to create their own games and felt more confident in their use of ICT. The activity gave rise to increased 'learning together' as pupils took on roles as experts and taught their peers. In particular, increases in motivation of previously disengaged pupils were noted and previously quiet children 'blossomed' using the *Kodu* program.

Increased motivation is also reported in a case study of UK secondary pupils (Passey, 2012), who created levels for the commercial game *Little Big Planet 2* (Sony, 2011) in after school clubs over a five month period. This research reports that pupils valued learning by doing, and being able to demonstrate their creativity. They also valued being given the opportunity to achieve professional outcomes and to engage with industry partners. Other indicators that pupils found the project motivating included increased engagement and enthusiasm, improved attendance, high levels of in-depth discussion, and the pursuit of high standards. They enjoyed learning dynamically, using multimodal material, rather than the static forms (texts, pages, diagrams) often predominant in school settings. Importantly, some pupils who were disengaged with learning became re-engaged through the project; less confident pupils became more communicative and self-esteem increased. Whilst this research is situated in the secondary phase it differs from the current study in that students were authoring levels

for a commercially produced computer game, rather than creating their own games from scratch. They did so largely in after-school contexts, over a much longer period, working in teams, with input from industry professionals. The focus was to develop cross-curricular 21st century skills and to foster interest in the computer games industry, rather than to learn about game design and programming in the context of the ICT curriculum. Yet despite these differences, the current study shares common ground with Passey's evaluation. Both studies identify a range of positive outcomes arising from complex, project-based activity and observe increased levels of confidence and engagement with learning for some pupils, as well as increased exposure to developing 'soft' skills, such as planning, working with others and problem-solving. In both studies pupils valued the extended time frame, although extended activities and attendance were identified as problematic within mainstream school settings. Passey's recommendations for increased use of collaborative tools and practices are also echoed in the present study.

These studies note that game authoring is motivating because pupils can pursue their individual interests, produce something of cultural significance and enjoy the playfulness inherent in the activity and learning in a dynamic, multimodal medium. However, because so much of the literature cross-references motivation, this review now turns to other research focuses.

2.5 Literacy and narrative development

A second body of work looks at how authoring computer games supports literacy and narrative development. Research surrounding the 'Adventure Author' project (Robertson and Nicholson, 2007) explores the development and use of software to enable young people to author computer games as a route to developing their narrative skills, since many children have difficulty in expressing themselves in writing and need to be given motivating and enabling opportunities to engage with story making (see Dillon, 2004; Good and Robertson, 2004; Robertson and Good, 2004; Robertson, 2004; Robertson and Good, 2005; Good and Robertson, 2006a; Good et al., 2007; Howland et al., 2008). This research describes how game authoring develops literacy and narrative skills via the production of non-linear narratives and branching dialogue. Related work focuses on the development of tools to support multimodal interactive writing, since it is difficult for young people to create compelling storylines without some method of keeping track of the development of their game narratives (Howland et al.,

2013). For these researchers, having to learn a programming language in order to author games distracts children from developing narrative skills. To address this, related research describes the development of *Script Cards* (Howland et al., 2006), software which enables pupils to create interactive stories, using graphics and natural language.

Game authoring also supports literacy development in terms of the range of activities pupils are engaged in when designing their games. These include seeing games as texts, learning to critique games and engaging with the 'paratexts' surrounding games (Buckingham and Burn, 2007a; O'Mara and Richards, 2012). Moreover, game authoring provides opportunities for alternative forms of narrative development outside the realm of text or speech (Carbonaro et al., 2008), where narratives are represented in dynamic, visual and interactive modes and pupils are engaged in 'multimodal literacy' practices (Burn, 2007; Beavis, 2013). Case study research using *MissionMaker* (Immersive Education, 2007) at secondary level finds that game design develops pupils' awareness of what constitutes narrative and provides new opportunities for the production of narrative in which children learn about the 'grammar' of games (Buckingham and Burn, 2007b; Burn, 2007). This incorporates a first or third person player point of view, the imperative mood in which instructions are given and the 'If…' clause, where much of the game play is based on conditionality. The authors note the affinity between computer games and oral narratives (e.g. folk and fairy tales) in terms of the episodic structure, the economies of health and magic, the character archetypes and the narrative roles they occupy. Additionally, game narratives involve the construction of rules; choices and interactions within the game may change the course of subsequent game play; life, health and score economies have impacts on the story. Pupils need to understand this 'grammar' when they play and create their own games, it is argued.

Other case study research explores how making games using *Kodu* (Microsoft Research, 2009) supports literacy development in middle school pupils (Tesk and Fristoe, 2010), and foregrounds how compositional practices and skills are enriched by authoring in a digital mode, because notions of user experience and audience are central to this process.

But game authoring is not just used as a context in which to develop narrative and literacy skills - it has benefits for other subjects too.

2.6 Game authoring across the curriculum

The breadth of learning experience that game authoring affords was identified by Kafai (Kafai, 1995) in a study in which Year 4 children made their own educational fraction games using the *Logo* programming language. During the six month project, children adopted many roles, as users, designers, writers of storylines, teachers (of fractions concepts), and programmers. They developed increased understanding of programming and mathematical concepts, as well as metacognitive skills in planning and monitoring their work. Kafai's research exemplifies constructionism in practice in the primary phase, however, the extended time frame of the research brings into question the applicability of its findings for current mainstream secondary ICT settings. Kafai's students spent 92 hours on programming and 20 hours on other activities related to the project. While Kafai observes that the long term involvement in the project was essential for students' learning (ibid.: 290), it would be difficult to integrate such an approach into the current UK ICT curriculum, which is typically allocated one hour a week at Key Stage 3.

Similar research was conducted by Idit Harel in her Instructional Software Design Project (Harel, 1991), where computer games were authored in *Logo* by 17 Year 4 pupils, over a four month period, to teach younger students about fractions. Findings showed that in creating their games, pupils achieved greater mastery of both *Logo* and fractions and that learning them simultaneously was more effective than learning either in isolation (Harel, 1991: 391). The constructionist philosophy underpinning the research was seen to facilitate personal engagement, the gradual evolution of different kinds of knowledge and the sharing of that knowledge with others. As with Kafai's work, the importance of learning over time is emphasised, although Harel acknowledges the 'extreme' environments established by the research project (Harel, 1991: 337), which involved 70 hours of work. This intensive approach carries through to her current work, which presents an online game design curriculum requiring a time commitment of between 40 and 120 hours (Harel Caperton et al., 2010). This sets it apart from the current study, which endeavours to introduce game authoring into the 'everyday' UK ICT curriculum, with its discontinuous and fragmentary provision.

More recent case study research in how making games supports the learning of mathematics (Ke, 2014) found that middle school pupils' dispositions towards the subject were significantly more positive after making games in *Scratch*, and they made better connections with everyday mathematical experiences, but that game

construction and narrative involved pupils more than the representation and integration of mathematical content.

Game authoring has also been used in the science classroom, in a study which adopted an experimental design to identify whether making games had any influence on Year 5 pupils' knowledge of nutrition concepts (Baytak et al., 2008). Although there was no significant difference in pre- and post-test scores, qualitative data indicated that understanding of nutritional concepts was conveyed in pupils' game narratives, although lack of programming skills limited the extent to which they could apply their knowledge.

While this study found that there were no significant differences in science knowledge before and after the game-making intervention, another empirical study (Yang and Chang, 2013) demonstrated significant improvements in understanding of science concepts when a biology topic was integrated with programming classes to create a computer game using *RPG Maker* (Enterbrain, 2005). The same tool was used to explore whether learning history at tertiary level could be enhanced through game authoring (Lim and Binti Md Sabri, 2013). Five students participated in a voluntary workshop (one hour a week over 8 weeks) to develop a historical role-playing game and findings from this research showed that game authoring promoted collaboration and engagement, generated different perspectives of historical events and gave students an alternative arena in which to present historical knowledge.

Despite the importance of the research reviewed in this section, its usefulness to the current study is only partial, since the dominant site for much of this research is the primary school. Although the research involves the creation of computer games, the emphasis is on the learning which is achieved in a range of curriculum subjects. In contrast, the focus of the current study is on how authoring computer games supports the learning of basic game design and programming concepts within the ICT curriculum. Furthermore, its interest in the collaborative and creative aspects of computer-based constructionist activities differs from the literature on computer supported collaborative learning (for example, see Luckin, 2010). Luckin's interest lies in how sociocultural theory can be extended in the design of educational technology and technology-rich learning activities. Building on Vygotsky's notion of the Zone of Proximal Development, Luckin refers to a Zone of Collaboration (Luckin, 2010: 28) which consists of interactions between collaborating individuals, and their more able partner (which includes technology), the resources available and the artefacts created.

Whereas the software she evaluates is designed to support collaboration in the primary phase between learners and their more able partners, *Game Maker*, the software used in the current study, was not designed for this purpose. Indeed its interface provided little support either for domain knowledge, or the development of metacognitive skills. Collaboration in the current study then, was supported, not so much by the software, but by pupils working with a partner, with peers, within the collective of the group, in the context of a complex, open-ended design task. In contrast, Luckin's research refers to software which supports collaborative learning in the completion of well-defined, closed tasks, which Luckin acknowledges is a limitation of the work (Luckin, 2010: 91).

2.7 Digital, game and media literacy

Other studies of computer game authoring view the activity as part of the media studies curriculum, where computer games are seen as 'new media' and the creation of computer games is a new, 'digital literacy' practice (Buckingham and Burn, 2007b; Peppler and Kafai, 2007b; Willett, 2007; Payton and Hague, 2010; Kafai and Peppler, 2011). The argument here is that computer games are an important cultural form and as such pupils need to develop a critical understanding of how this medium works by analysing game texts and exploring their appeal. They also need to be given the opportunity to 'write' games as well as 'read' them, since creating games allows for a more profound and engaging form of learning than analysis alone (Buckingham and Burn, 2007b).

Research conducted from a media studies perspective is exemplified by the literature surrounding the 'Making Games' project, which describes the development of the 3D game authoring tool, *MissionMaker* and its use in secondary schools (Buckingham and Burn, 2007a; Buckingham and Burn, 2007b; Pelletier, 2007). The software enables users to create graphic-rich 3D games, which are populated with ready-made characters, objects and environments. Opportunities for developing understanding of the design of games as media artefacts are uppermost in this software, but opportunities for developing understanding of programming concepts are limited to 'If … then' constructs. In these respects, the research relating to *MissionMaker* is of only partial relevance to the current study. Its focus on computer games as a new cultural form, a medium that young people should learn to critically evaluate and understand, important though it is, is of less concern to the ICT curriculum and the research focus of this study. From an ICT perspective, authoring computer games enables pupils to

develop programming skills, to plan and create interactivity, to consider human computer interface design and other ICT curriculum-related learning (Dalal et al., 2012). Authoring computer games from these perspectives is missing from the 'media literacy' and *MissionMaker* research.

A further area of interest from a media studies perspective is that computer games are 'popular' texts, which merit study in an inclusive curriculum. Thus game authoring might be more accessible to some children than traditional forms of textual study (Burn, 2007). Viewing the playing and making of games in this light is also part of a 'democratisation [of learning]' - where students' out of school cultures are recognised as valid and worthy of consideration in the school curriculum (Buckingham, 2003; Beavis, 2013). Similar arguments in terms of democratising the creation of digital media and providing alternative pathways into participatory culture are also advanced in other studies (Peppler and Kafai, 2007a; Resnick et al., 2009a; Williamson, 2009). The current research acknowledges the importance of such perspectives, but is more concerned with the ICT-related learning which arises out of programming interactive media.

Other researchers have introduced the notion of 'game literacy' as a subset of media literacy (Burn, 2007; Salen, 2007; Zimmerman, 2009). According to this perspective, developing 'game literacy' is important because children learn to view games as designed systems (Salen, 2007) and become *systems literate* - they learn to understand games as dynamic sets of parts with complex interrelationships, see the structures that underlie them, and gain awareness of how these structures function (Zimmerman, 2009). The idea here is that being able to successfully understand, navigate, modify, and design systems is an important 21st century skill (Zimmerman, 2009).

According to Pelletier (Pelletier, 2005), developing game literacy involves studying systems in a different sense, in so far as games can be analysed as semiotic systems, sets of signs which can be 'read' and 'written'. Making games involves more open-ended and conscious manipulation of game-based semiotic resources than is achieved through game play alone. It is a means to develop understanding of media as cultural phenomena and to gain an awareness of the practical skills and creative abilities involved in media production. In creating games pupils use technologies more productively, and are able to participate in and contribute to media culture (Pelletier, 2005).

For Sanford and Madill, authoring computer games develops students' 'operational literacy' (Sanford and Madill, 2007b), which incorporates both software skills and the ability to understand the conceptual content and applications of a particular program (ibid.). Their study examined the 'new literacy' practices that secondary-aged boys engaged in when making games with *Stagecast Creator* (Tesler et al., 1997), out of school over a nine week period, and observed operational literacy to be widely practised (Sanford and Madill, 2007b). This competence in the skills, processes and techniques involved in making a computer game included understanding user input mechanisms (such as the use of a controller, mouse or keyboard), reading visual instructions, using and adapting semiotic systems, creating icons to communicate with players, and using technological language and the wider discourse of computer games.

From the perspective of these researchers, authoring games is important because it develops creative and critical practice in the realm of digital media, and brings students into contact with a range of new literacies arising from this. For other researchers, the focus is less about the product or the literacy and practical skills developed in its creation, and more about the design process involved, which is discussed in the next section.

2.8 Learning by design

Making computer games involves pupils in a design process, and foregrounds the importance of 'learning by design' (Kafai, 1995; Kafai and Resnick, 1996a; Kafai, 2006b; Peppler and Kafai, 2010; Ke, 2014). According to Kafai pupils have little experience in following the design process from beginning to end, which involves researching, planning, problem-solving, dealing with time constraints, modifying expectations and bringing everything together (Kafai, 1996: 71), because conventional school assignments rarely give pupils the opportunity to spend an extended period of time on complex projects (ibid.). For Kafai learning by design is important because it helps young people to learn how to learn. Since there is no single solution to the design problems involved in making a computer game, pupils can choose how they approach the task: their designs can emerge in the process of being created, or they can plan, implement and test their games iteratively. In following the design process, the pupils in her study developed a range of strategies to deal with the complexity of the game-making activity (they broke program code down into procedures and sub-procedures and re-used procedures that worked), but importantly, they needed a complex

programming project for it to make sense for them to do so. Kafai's later research in learning by design and making games describes how strengthening the notion of audience throughout the design process, by tasking pupils to design tangible game controllers for their games, and providing authentic audiences to showcase their work, encourages pupils to consider usability, functionality and player experience and to respond to user feedback, all important aspects of the design process (Davis et al., 2013).

The importance of design as an educational imperative is also recognised in more recent studies (Salen, 2007; Hayes and Games, 2008). Key research from this perspective centres on the development of *Gamestar Mechanic* (Games, 2008a), an online, multiplayer role-playing game, designed to develop '21st century literacy skills' by teaching the language and principles of game design. Learners develop a 'game designer discourse' (Games, 2008a) *through* the design and play of computer games. Game authoring is valued as a design practice because it encourages systemic thinking, specialist language and literacy skills, computational literacies and software design skills. However, current implementations of the *Gamestar Mechanic* curriculum (E-line Media, 2013) are delivered in schools over a semester (50+ lessons) and would be difficult to incorporate in the time available for ICT in many UK secondary schools.

Since it is difficult to learn about game design within the structure of the conventional secondary school timetable, extended programmes requiring a substantial commitment in terms of time (6-8 hours a week) have been developed to remedy this. *Globaloria* (Harel Caperton et al., 2006) offers an online environment in which students use social media to support their learning as they create web-games, using *Adobe Flash* (Adobe Systems, 2007). The programme enables students to 'learn to be' as they participate in a "networked, software design-based learning community" (Harel Caperton, 2010: 6). Research surrounding *Globaloria* identifies six 'contemporary learning abilities' afforded by game authoring in this context, which include making an original game, learning to project manage game production, publishing digital media, developing social learning skills, and learning to research from and evaluate web sites and web applications. These competencies are important aspects of game media literacy and necessary for effective learning and working in today's technology-driven landscape and global workplace (Harel Caperton, 2010).

Other research which focuses on learning by design seeks to support the challenges of making a computer game by developing software to scaffold the process (Robertson

and Nicholson, 2007). Game design problems are often open-ended, change over time and have multiple solutions, which can pose difficulties for novice game makers. This research looks at how the development of a 'Designer's Notebook' tool can support learners in an out-of-school workshop manage the 'complex process of design' and investigates the meta-cognitive skills which young people develop in the process of game design, including planning, reflection and self-organisation.

The importance of providing support for learning-by-design projects is acknowledged more recently (Ahmadi and Jazayeri, 2014) with the development of *AgentWeb*, an online game design environment which incorporates integrated learning resources, including video tutorials, synchronous communication tools such as chatrooms and instant messaging, and asynchronous communication via integration with Facebook and forum comments. This research suggests that such systems foster cooperative and collaborative learning and may offer effective models to support learners engaged with complex game design projects both within and without school settings.

According to these researchers, making computer games is an important design activity which gives pupils a real opportunity to develop as learners and to build important 21st century digital literacy competences within the context of a complex design project.

2.9 Game authoring and computer programming

Previous sections in this literature review have illustrated how game authoring has been researched from a variety of perspectives. Making computer games emerges as a motivating context for learning about a range of subjects at primary level as well as supporting the development of literacy and narrative skills; at secondary level it is an important context for developing game and media literacy. In both phases, pupils enjoy the opportunities it offers to be playful and creative and this brings positive effects in terms of increased commitment to learning.

However, beyond developing generic learning and digital literacy skills or enhancing knowledge and understanding in other subjects, there is another body of research which looks at how game authoring introduces pupils to programming concepts and practices, and it is to this area that we now turn.

In ICT education the notion that pupils should be engaged in activities which go beyond the presentation of information to the development of systems which process data and 'make things happen' (QCA, 2007b) is strongly advocated (Peyton Jones et al., 2007). The newly-minted Programme of Study for Computing (DfE, 2013c) transforms that idea into educational policy and is the culmination of recent combined lobbying on the part of industry and educationalists to revamp a reportedly ailing ICT curriculum and bring computer programming and computational thinking centre stage (see CAS, 2008a; Livingstone and Hope, 2011; Schmidt, 2011; Furber, 2012).

Making computer games fits neatly into the picture, since all game authoring requires some form of programming. Accordingly, game authoring is being increasingly used from primary to tertiary levels as a motivating and contemporary scenario to support the learning and teaching of this aspect of computer science (see Denner et al., 2005; Sanford and Madill, 2007a; Hayes and Games, 2008; Repenning and Ioannidou, 2008; OCR, 2009b; OCR, 2011; AQA, 2012b; AQA, 2012c; Edexcel, 2012a; OCR, 2012a).

However, learning to program is a difficult task (du Boulay, 1986; Soloway and Spohrer, 1989; Jenkins, 2002; Robins et al., 2003; Dagdilelis et al., 2004; Lahtinen et al., 2005; Parsons and Haden, 2007; Hernandez et al., 2010; Saeli et al., 2011). Du Boulay identifies 5 areas which commonly cause problems: understanding what programming is for; understanding what is going on inside the machine; learning to use notation (syntax, semantics) and structures (loops, conditions), and learning how to specify, develop, test and debug programs (du Boulay, 1986: 57).

Difficulties in learning to program also arise because pupils expect that the computer will interpret what they mean rather than do what they write (Pea, 1986), and because novices don't understand the specialist meanings everyday words have in computing or realise the level of detail required in writing programs (du Boulay, 1986: 62).

Other areas of difficulty in learning to program are that pupils find it difficult to break down problems into more manageable sub-problems and fail to pre-plan the necessary components of the program (Pea, 1983; Perkins, 1986). Another common source of error is that pupils merge processes when they should be implemented separately (Spohrer and Soloway, 1989). In fact, basic program planning emerges as the major source of difficulty for novice programmers (Robins et al., 2003). In particular pupils need more instruction on 'how to put the pieces together' (Soloway and Spohrer, 1989: 412).

Taking a different perspective, Perkins et al. (1986) suggest that difficulties in programming arise from pupils' dispositions, behaviour and attitudes to learning. Their research suggests that many pupils disengage from tasks when errors arise and delete rather than try to fix errors, or avoid dealing with mistakes and turn their attention instead to a different task. Others neglect to track what their programs are doing by reading code as they write it, or try to repair programs by tinkering haphazardly with code without thinking about the problem or its solution. While these forms of impulsive, unreflective programming can be remedied by teaching pupils to read/track their code and check their work, pupils commonly neglect to do this and rarely do so without prompting (Perkins, 1986: 270). According to this research, pupils' experiences of writing programs could be improved if better learning practices were encouraged.

Difficulties also arise because programming has a relatively undeveloped pedagogy, pupils may only learn it for one hour a week, many teachers are new to programming themselves (Perkins, 1986: 262) and non-motivating contexts are often used to teach programming concepts (Good et al., 2007).

Because of the difficulties learners have with learning to program (Kelleher and Pausch, 2005), visual programming languages have been developed to make programming more accessible (Baldwin and Kuljis, 2000). Such languages use graphical representations (such as flow charts and icons) of program elements as the constituents of a program, instead of text. These graphical elements are combined in a drag and drop environment to create the program code, and because of their accessibility, visual languages are preferred over textual systems as a means to introduce programming in primary and secondary schools (Murnane, 2010).

This section describes visual programming languages commonly used in UK schools which are featured in research projects relating to the teaching of programming and/or game authoring and then goes on to survey the research surrounding *Game Maker*, the game authoring tool used in this study.

Scratch

Scratch (Resnick et al., 2003) is widely used in UK primary and secondary schools to create 2D games and animations (see Burtoft et al., 2008; Scott, 2011; The LEAD Project, 2012). Pupils select graphical blocks to compile behaviours for game objects, which when 'run' produce a visual output. These blocks resemble jigsaw pieces, and can only be combined in syntactically correct formations (see Figure 4).

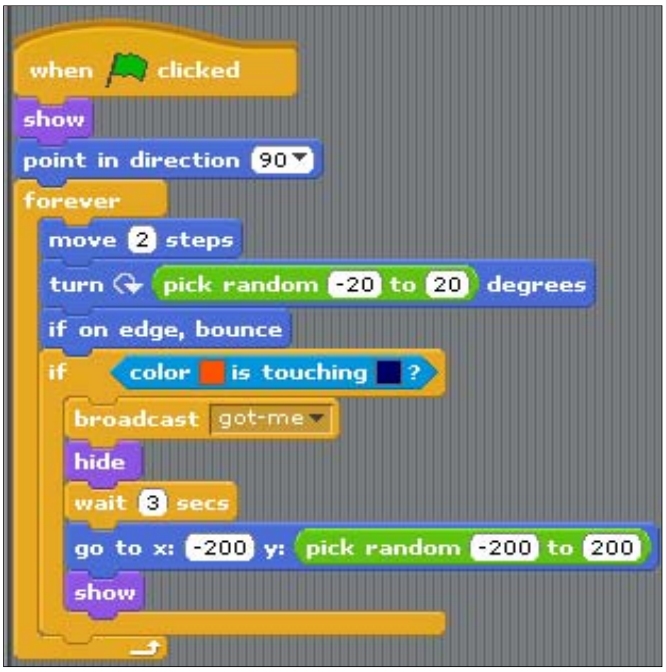


Figure 4: A script in Scratch's colour coded blocks

For the researchers involved in the development of *Scratch,* young people need to learn to 'write' as well as 'read' the full range of digital media. Digital media creation is seen as an important 21st century skill and an avenue for creative self-expression. Accordingly, the program has spawned a large body of research in digital media production, although much of this research is conducted in the context of after school programmes for 'underserved' communities, where the focus is on programming artefacts in the 'media arts', rather than game authoring alone (see Peppler and Kafai, 2005; Peppler and Kafai, 2007b; Maloney et al., 2008; Kafai and Peppler, 2011).

Significant research interest has been shown in how creating games with *Scratch* supports the learning of programming concepts (Peppler and Kafai, 2005; Maloney et al., 2008; Li, 2010; McInerney, 2010; Baytak and Land, 2011a; Adams and Webster, 2012). One such study (Adams and Webster, 2012) analysed 300 *Scratch* projects created by middle school summer camp children and found that making games is a better scenario than creating stories and animations for teaching students about variables and conditionals in particular, as well as other important programming concepts (loops, Boolean expressions).

One of the strengths of using *Scratch* to learn about programming is that it is accessible to novice programmers and teachers with non-computing backgrounds alike (Maloney et al., 2008; McInerney 2010). An analysis of 536 *Scratch* game projects, created by 8-18 year olds in an extended US after-school programme, documents the

learning of key programming concepts even in the absence of instructional interventions or experienced mentors. Programming concepts evidenced included sequence, threads, loops, conditional statements, Boolean logic, variables, and random numbers, although some of these concepts were not easily discovered alone (Maloney et al., 2008).

Another investigation of games made in *Scratch* (Baytak and Land, 2011b) found that Year 5 pupils (n=10) learned to use a range of programming constructs in their games, although they needed some teacher help and rarely used complex commands. Wilson et al.'s *Scratch* research (2012) found that mixed gender pairs achieved the highest mean score for use of programming concepts and produced the most functional games.

Other work focuses on the social dimension of using *Scratch* and how its online community supports young people as designers of interactive media (Brennan and Resnick, 2013) and evaluates the competing roles of structure versus agency in digital media creation in and out of school settings (Brennan, 2013b).

However one study suggests that the exploratory learning promoted by *Scratch* might actually be detrimental to learning programming (Meerbaum-Salant et al., 2010), observing that using *Scratch* engenders habits of programming which are not helpful, including a bottom-up approach to program development and a tendency to decompose programs into too many scripts. These behaviours are problematic because they are at odds with accepted practice in computer science, which encourages the planning and design of programs and the use of programming constructs to structure programs before they are implemented.

Kodu

Using a different paradigm, Microsoft *Kodu* (Microsoft Research, 2009) offers a tile-based visual language to enable children to create 3D games (see Figure 5). *Kodu* differs from other educational programming environments in that it runs on the Xbox games console as well as desktop computers and was designed for children who have never known a world without visual user-interfaces and game consoles (Coy, 2013).

In one quantitative study (Stolee and Fristoe, 2011) researchers counted the number of programming concepts used in 346 games shared on the *Kodu Xbox Live* community website and found that users were able to express several computer science concepts

Figure 5: Kodu's 'When…do' condition/action rule tiles

including variables, conditions, Boolean logic, objects and the flow of control. Related research (Fristoe et al., 2011) describes the development of a set of gender-inclusive game mechanics added to *Kodu* to enable girls to create games based on features such as dynamic relationships, social interactions and storytelling. The research describes after-school sessions in which self-selected middle school girls evaluate these game extensions, and finds that although girls liked these new capabilities, few used them in the games they made because of their complexity.

Alice

Alice (Cooper et al., 1999) offers a visual environment to teach students programming concepts as they create 3D games, animations, and stories (see Figure 6). Recent research (Werner et al., 2012a) measured the frequency of programming constructs used in 231 games created in *Alice* by middle school pupils (n=325) in out-of-school classes and in-school electives and found that they learned about sequential, conditional and parallel execution and that nearly a third of games contained conditionals and variables, nearly half contained functions and 85% used events.

Another study (Kelleher and Pausch, 2007), found *Storytelling Alice* (Kelleher, 2006) to be a motivational means to learn about programming for middle school girls (n=23). In a 4 hour game project, all managed to create a sequential program, 87% included multiple methods and several used loops and parameters; however, conditionals and variables were not so straightforward and targeted assignments were needed to deliver these concepts successfully. In a longer, 20 hour out-of-school programme, the games produced showed that middle school pupils were able to positively engage with computing concepts such as algorithmic thinking, programming, modelling and abstraction (Werner et al., 2009).

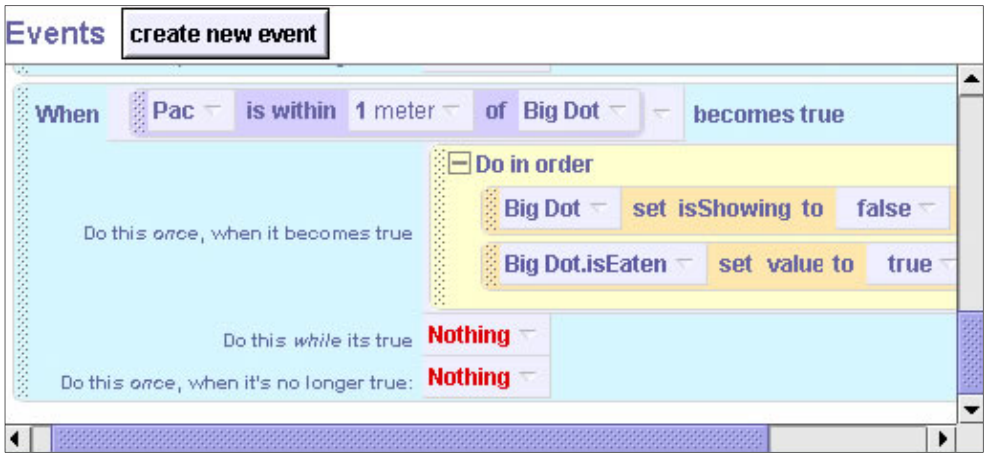


Figure 6: Alice's programming panel

This section has given a flavour of the research surrounding 3 visual programming environments used to teach programming at Key Stage 3 in the UK. Many of these studies feature intensive, out-of-school programmes, with primary or middle school pupils, and are rarely located in the UK Key Stage 3 ICT context. In contrast, the present study explores how basic programming concepts are introduced via game authoring as part of the 'everyday' formal ICT curriculum.

The next section surveys the literature surrounding *Game Maker*, the game authoring tool selected for this study.

2.10 Game Maker

Research conducted in the United States reports how *Game Maker* has been used to introduce computing concepts associated with game implementation, such as objects, conditional statements and loops, and to encourage uptake of computer science courses at tertiary level (Chamillard, 2006; Dalal et al., 2009). In this context, *Game Maker's* graphical interface was found to be useful for introducing programming concepts first, before transitioning to its textual language (Hernandez et al., 2010). Using this staged approach resulted in improved student performance in programming assessments over previous years. Dalal et al. (2012) come to similar conclusions in their research, which uses *Game Maker* for rapid game creation as an alternative approach to teaching programming concepts, suggesting that it complements the use of textual programming languages in computer science education.

Other US research describes how *Game Maker* was used in a 4 day summer camp for 18 pupils in Years 6-12 (Guimaraes and Murray, 2008). The study identifies strategies

to increase student engagement and learning, and recommends a 'play, modify, create' model for game authoring. The authors stress the importance of allowing students to practice reading and modifying the code in sample games before they engage in any code creation themselves, noting that students are usually given the task of creating programs before they have learned how to read and understand them (Guimaraes and Murray, 2008).

More recent US research has investigated how *Game Maker* can be used to support the learning of computer science concepts, as well as addressing learning objectives in other subjects such as mathematics and English (Doran et al., 2012). This study describes the evolution of a 10 week out-of-school programme and focuses on the pedagogy of game authoring. The authors recommend giving pupils time to plan their program segments and write the pseudocode for them before they implement their games and including 'guided errors' to increase pupils' debugging abilities, noting that pupils responded best when they were encouraged to make mistakes rather than avoid them. The authors also describe errors they made in the programme, which included giving pupils free choice in their game designs; this made delivering the programme more difficult because teachers had to provide different instruction for each pair. Pupils' unrealistic expectations of the games they could produce also caused problems. In subsequent iterations the programme was modified to include more structure and more development time, and clarified the sorts of games pupils could realistically create. The task was recast as creating a prototype, focusing on game mechanics rather than graphics and aesthetics. This allowed for more structured, targeted lessons to be delivered (Doran et al., 2012). These lessons learned closely reflect similar experiences in the current research.

Game Maker also features in research which investigates how game authoring can enhance the learning of science in the primary phase (Baytak et al., 2008; Baytak and Land, 2010; Baytak et al., 2011). This case study follows Year 5 pupils (n=10) who make games to teach younger pupils about nutrition (Baytak et al., 2011). Findings show that making games allowed pupils to represent their knowledge about nutrition in concrete and personally meaningful ways, and that the activity was engaging and motivating for pupils (Baytak and Land, 2010). However, there were challenges, notably with implementing game designs with limited programming skills and customising graphics (Baytak et al., 2011). Another problem was that the process of creating games dominated classroom activity, to the extent that the topic focus - learning about nutrition, was marginalised (Baytak et al., 2011). Indeed there was no

significant difference in pre-test and post-test scores of pupils' knowledge (Baytak et al., 2008). While the report observes that pupils used increasing numbers of actions in their games as the project progressed (Baytak et al., 2008) there is no reference to learning about programming beyond this.

Other research studies which refer to *Game Maker* do not investigate the learning in programming that is achieved when pupils create games, or its utility in the ICT curriculum. Rather, their main focus is on how the program has been used to enhance particular aspects of learning, such as creativity (Eow et al., 2010), or digital literacy and multi-literacies (Sanford and Madill, 2007a; Beavis and O'Mara, 2010; Beavis et al., 2012; O'Mara and Richards, 2012) or multimedia design (Beavis et al., 2012) or how the program has been used as a motivation for learning in other subjects (Fluck and Meijers, 2006; Baytak et al., 2008), or how making games enhances collaborative working practices and promotes social constructivist learning environments (Madill and Sanford, 2009). There are few published studies which focus on *Game Maker* and how it is used to teach programming or game authoring concepts in the UK secondary ICT curriculum (Hayes and Games, 2008; Daly, 2009).

2.11 Summary

In this chapter I have set the context for the study by reviewing the literature relating to computer game authoring, organising the various research focuses into themes, although there is often a crossover between these. According to this literature, game authoring motivates and engages learners in a range of contexts, and has been used as an alternative approach to learning subject content across the curriculum, to support literacy and narrative development, and to provide an engaging and accessible introduction to computer programming. It is apparent that there is significant interest in the area of computer game authoring, from a range of perspectives, but that there is a gap in the literature relating to the study of how computer game authoring is used in the UK Key Stage 3 ICT curriculum to support the learning and teaching of basic programming concepts. The present research adds to knowledge in this respect and seeks to address the following research questions, which explore areas of interest not widely focussed on in other studies:

1. Since game authoring was a new context for learning for the pupils in this study, and taking into account the recent debates surrounding the ICT curriculum (see Chapter 1), this research seeks to explore pupils'

perceptions of the process and outcomes of their encounters with such new curricula. Did they enjoy the activity, what did they make of the resources and software provided and was working in pairs on an extended, open-ended task, in which they had some control over their own intellectual activity and learned in the process of making a digital artefact, an effective way of working?

What are pupils' perceptions about the process and outcomes of their learning during a constructionist-designed game authoring activity?

2. As noted in Chapters 1 and 2, *Game Maker* is widely used in UK secondary schools, but under-researched in that context. Does this visual programming tool, with its graphical interface and integrated development environment provide an accessible and effective means of learning basic programming concepts for Year 9 pupils?

   How does computer game authoring using Game Maker support the learning of basic programming concepts and practices?

3. Chapter 1 provided a background for why engaging and accessible contexts for learning to program are important for the new Computing curriculum. Arising out of this is the need for a greater understanding of the difficulties pupils have with designing and programming computer games.

   What difficulties do pupils have with game authoring (game design and game programming)?

4. Several studies reviewed in Chapter 2 identify a range of positive outcomes which occur when pupils author computer games in primary and out-of-school contexts. As an extension of this, the current research explores what value there might be to pupils in a secondary mainstream setting in authoring computer games, beyond the learning of curriculum content. Does the activity generate positive outcomes in terms of affect or in changed attitudes to learning or in respect of pupils' relationships with technology?

   What affective value is there to pupils in authoring computer games?

In the next chapter I outline the conceptual framework which underpins the research and analysis.

Chapter 3 The theoretical framework

3.1 Introduction

This thesis builds its conceptual and analytic frame from the work of Seymour Papert and the learning theory of constructionism which he first conceived some 30 years ago (Papert, 1980b, 1993). Throughout the intervening period numerous researchers have been influenced by Papert's ideas (e.g. Edith Ackermann, Karen Brennan, Idit Harel, Celia Hoyles, Yasmin Kafai, Ken Kahn, Richard Noss, Mitchel Resnick), but rather than promoting new constructs, they refer to Papert's work and apply his principles in their own later research. Proceedings of the biennial Constructionism conference contain papers which describe the use of new computational tools and refer widely to constructionism, but they do not significantly add to the theory. For example, Kafai and Burke's paper, Mindstorms 2 (Kafai and Burke, 2014) argues that 'computational participation' increasingly embraces social connectivity and that this has changed practices from writing code to creating applications, from composing from scratch to remixing others' work, from designing tools to facilitating communities and from screen-based environments to tangibles. This work presents a development of constructionist *practice*, not of the underlying theory, acknowledging instead that Papert's vision is at last coming to fruition (Kafai and Burke, 2014).

This chapter therefore outlines the key characteristics of Papert's original theory, acknowledging its constructivist roots in Jean Piaget's research in epistemology (Piaget and Inhelder, 1969; Piaget, 1972; Piaget, 1973). Papert worked with Piaget from 1958 till 1963 and later acknowledged that Piaget's ideas were at the centre of the concerns of his book, *'Mindstorms'* (Papert, 1980b: 217), in which he advanced his ideas of constructionism and presented *Logo* (Papert et al., 1967) as an archetypal constructionist learning environment. It is beyond the scope of this study to refer to Piaget's theories in detail. Instead, I show how Papert has built on a number of key ideas within a computing framework to develop the learning theory of constructionism.

Papert defined constructionism, his "personal reconstruction of constructivism" (Papert, 1993: 143), in a proposal to the National Science Foundation, as follows:

> The word constructionism is a mnemonic for two aspects of the theory of science education underlying this project. From constructivist theories of psychology we take a view of learning as a reconstruction rather than as a transmission of knowledge. Then we extend the idea of manipulative materials to the idea that learning is most effective when part of an activity the learner experiences is constructing a meaningful product (Papert, 1986).

His purpose here was to extend the constructivist learning theory that children build their own understandings, by suggesting that they do so more effectively when they are able to actively construct artefacts that have some resonance for them. The process of making a product is one aspect of constructionism; the idea that such products should also have cultural and personal significance is another defining factor.

Papert later promoted the term to describe the theory of learning which grew out of his work in developing the *Logo* programming language. He chose the word to evoke and to synthesise the psychological term 'constructivism' and the image of a construction set (such as *Lego* or *Meccano*). The key idea is that "building knowledge structures ('in the head') goes especially well when the subject is engaged in building material structures ('in the world') as children do with construction sets" (Papert, 1991a: xi). Significantly, constructionism extends the connotation of 'construction set' to include programming languages - 'sets' from which programs can be made (Papert, 1993: 142) and much of the research surrounding the evolution of constructionism is related to computer-based learning environments which involve some sort of programming activity (see Harel, 1991; Harel and Papert, 1991a; Kafai and Resnick, 1996a).

This research was conducted by Papert and the Epistemology and Learning Research Group at the Massachusetts Institute of Technology (MIT) Media Laboratory, from 1985 onwards. The group's work sought to advance the idea that "computational technology would give rise to a new science of expressive media" (Papert, 1991b: ix). Crucially, the computer would not be used solely to deliver instructional material - children would instead learn how to use computers to express their understanding of geometry, write programs, create graphics or make a computer game, for example. Children would learn to be "producers instead of consumers" of educational software (Papert 1993: 107). This idea was important at a time when computers had just begun to be widely available in schools and discussions of how computers should be used and how they would change the learning landscape were then, as now, widespread. Papert's vision was that the computer would not be used as a tool to reinforce traditional methods of

teaching and learning, but that it would radically transform them. Constructionist computer-based learning environments would enable children to interact with powerful ideas (Papert, 1980b), and give them 'good things to do' so that they can 'learn by doing' better than they could before (Papert, 1980s).

Researchers working alongside Papert at MIT have published widely about constructionism in practice in the United States, both within formal educational settings, and in out-of-school programmes, with a particular focus on exploring how computer programming activities promote learning in mathematics and science education in the primary phase (see Harel, 1991; Harel and Papert, 1991a; Kafai, 1995; Kafai and Resnick, 1996a), and more recently, how they enable learners to develop digital literacy skills in the creation of multimedia artefacts (Kafai et al., 2009b; Resnick et al., 2009b). The current study builds on this work by exploring how the constructionist activity of authoring a computer game supports the development of basic computer programming skills within the mainstream UK Key Stage 3 ICT context.

3.2 The 8 big ideas of constructionism

As the concept of constructionism developed, Papert crystallised its key characteristics into '8 big ideas' (Papert, 1999 in Stager, 2007) as shown in Figure 7.

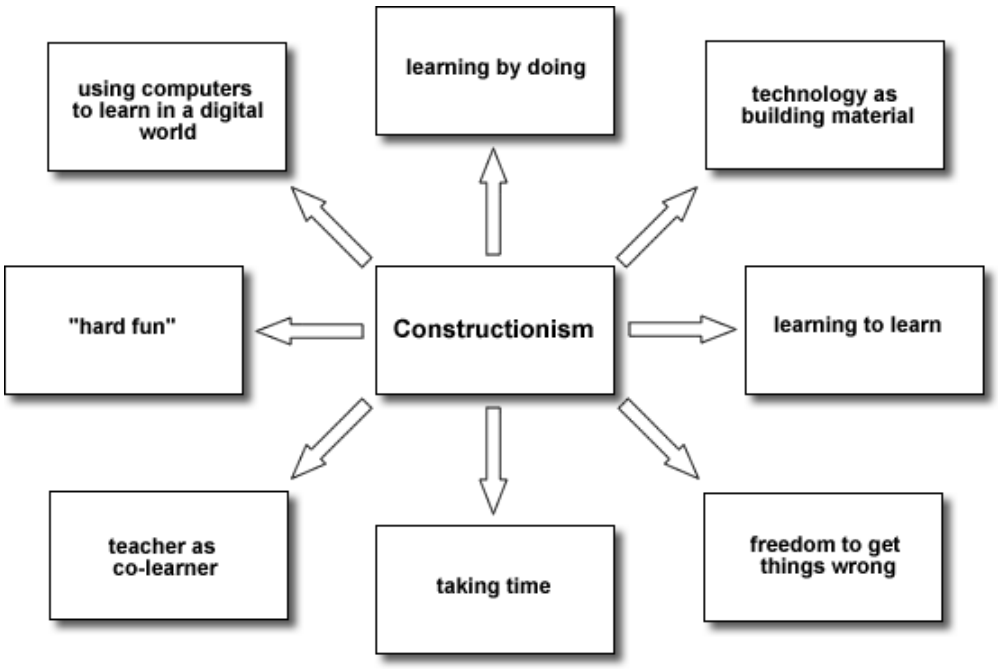


Figure 7: The eight big ideas of constructionism

These ideas were used as the guiding principles in the foundation of the Constructionist Learning Lab[3], the first learning environment built entirely from the ground-up to support constructionism (Stager, 2007).

In the following section Papert's eight big ideas are introduced to present constructionism as a set of principles which together articulate the theory of learning which underpins the present thesis.

1. Learning by doing

"The first big idea is learning by doing. We all learn better when learning is part of doing something we find really interesting. We learn best of all when we use what we learn to make something we really want" (Papert, 1999a).

At its simplest, constructionism can be defined as "learning-by-making" (Papert, 1991b), although as Papert observes, constructionism is best conceived by actively building an understanding of it, rather than accepting a definition of it. Moreover, the theory and applications of constructionism have a much wider reach than a simple definition can encapsulate (Papert, 1991b). While constructionism shares constructivist views of learning as "'building knowledge structures' through progressive internalization of actions …" (Papert, 1991b: 1), it then adds the idea that this happens "especially felicitously when the learner is engaged in the construction of something external or at least sharable … a sand castle, a machine, a computer program, a book" (Papert, 1990b: 3).

The key idea here is that making something tangible supports learners in thinking about their own thinking. In computer-based constructionist learning activities, children learn to program, not for its own sake, but to create a personally meaningful artefact, such as a computer game, or a piece of software to teach others about a school topic, or a program to control a robot, or an animation. Importantly, using the computer as a constructive medium in this way is empowering and dispels the idea that technology is something that only 'other people' can understand and make (Papert, 1996a: 48). In the process of making digital artefacts, it is argued, children learn *how* to learn, and how to think computationally. In short, constructionist activities which allow children to make a product offer more authentic outcomes for learning than focussing on "the

[3] The Constructionist Learning Lab, designed by the Seymour Papert Institute, was a pilot of an alternative learning environment commissioned by Maine's State Education Department as an educational intervention for the Maine Youth Center, a residential facility for young offenders (Cavallo et al., 2004).

acquisition of knowledge and facts without a context in which they can be immediately used and understood" (Papert, 1986: 8).

This vision for learning with computers challenges what Papert saw as the reductive, instructionist uses to which computers were predominantly being put in schools, as exemplified by computer aided instruction (Papert, 1980b, 1993). He hoped instead that computers would enhance learning and transform education, "provide rich soil for the growth of intuitions and concepts for thinking, learning and playing" (Papert, 1970: vii-3), "add new degrees of freedom to what children learn and how they learn it" (Papert, 1980a: 209).

2. Technology as building material

"If you can use technology to make things you can make a lot more interesting things. And you can learn a lot more by making them" (Papert, 1999a).

To realise the full potential of computers to enhance learning requires appropriate computer-based learning environments and activities. An important part of constructionist philosophy was the development of such environments. Papert developed the *Logo* programming language as a tool to support an alternative, constructionist method for learning mathematics. This language was used to give commands to a 'turtle', initially a physical floor robot and later, a virtual onscreen object. The turtle gave a visual output to commands given to it and enabled users to create turtle graphics and later, games and animations. With environments such as *Logo*, the computer was used to build knowledge at the same time as building a computational artefact. The *Logo* turtle was designed as a "constructed, computational 'object to think with'" (Papert, 1980b: 11) and supports learning because it offers possibilities for children to identify personally with the computer. For example, a *Logo* program runs a sequence of operations on the child's behalf and the child can identify with the movement of the turtle on the screen. To control the movement of the turtle children use mathematical concepts, such as angles, distance, size, and rotation in a meaningful context, thus "knowledge is acquired for a recognisable personal purpose" (Papert, 1980b: 21). Papert referred to this as 'knowledge in use' (Papert, 1993: 63) and distinguished it from knowledge to be memorised and regurgitated in tests and examinations, or outcomes which are otherwise dissociated from the process of learning.

Throughout his publications, Papert was eager to characterise the computer as a means to enable exploratory learning. For him the role of the computer was to give children a sense of empowerment, to enable them to do more than they could do before and to allow them to learn certain concepts in a natural way, through play (Papert, 1984). For Papert, the computer was a 'mudpie' (Papert, 1984), 'material', that can be 'messed around with' just as paints, clay, crayons, wood, rulers and other materials can (Franz and Papert, 1988). In his view, educational computing should be active, exploratory, student-directed, and computers should be used as a constructive medium to enable children to do things and make things (Papert, 1998a), not reduced to tools of the 'information age', their primary use to provide information or access to information.

According to Papert, computer-based constructionist learning environments can help children build more advanced intellectual structures, but in order for them to do so, more emphasis needs to be put on making such environments available in our culture. Teachers need to make available those cultural materials which are relevant to intellectual development (Papert, 1980b: 32) and exploit 'cultural trends' in their educational interventions (Papert, 1980b: 181). In terms of the focus of the current research, one of those cultural trends is computer games.

3. Hard Fun

"We learn best and we work best if we enjoy what we are doing. But fun and enjoying doesn't mean 'easy'. The best fun is hard fun" (Papert, 1999a).

Constructionism is not solely concerned with learning in cognitive terms but also in terms of affect (Papert, 1986; Kafai and Resnick, 1996b). 'Affect' in this context refers to the relationship between learners and the work they do and the level of engagement which arises from it. The key idea is that learners become more intellectually engaged when they are given activities and projects which are personally meaningful to them (Kafai and Resnick, 1996a: 2).

Whilst learners will be more engaged with projects that have some resonance for them, they will also enjoy it more if what they are doing is challenging. Papert's notion of 'hard fun' was first articulated in his book *The Connected Family* (Papert, 1996a), inspired by a child who, having used computers for the first time, described the experience as fun, but really hard (Papert, 1996a: 53). Papert interpreted this to mean that it was fun *because* it was hard, that challenging activities give rise to enjoyable learning

experiences. But these challenging activities have to be personally and culturally relevant. They must connect with children's interests and with areas of knowledge and skills which are needed in the future (Papert, 2002: A7). Papert suggests that the concept of 'hard fun' is "widely present in children's thinking" (Papert, 1996: 53), particularly in the context of computer games, since the best games involve children in some "very hard learning" (Papert, 1998b: 87), but children are not daunted by challenging activities as long as they are also interesting (Papert, 1998b: 88).

Papert later describes 'hard fun' as a fundamental principle of learning (Papert, 1996a: 52) and contrasts it with "'touchy feely … make it fun, make it easy' approaches to education" (Papert, 2002: A7) and the marketing messages of educational software companies which claim to make learning 'easy' (Papert, 1998b: 87). For Papert, that should not be the goal of education or curriculum design. Indeed, "[his] whole career in education has been devoted to finding kinds of work that will harness the passion of the learner to the hard work needed to master difficult material" (Papert, 2002: A7).

The difficult material that Papert refers to includes programming and mathematics, which were the areas of learning most involved in early constructionist activities (see Papert, 1980b, 1993). In creating *Logo*, Papert sought to make those subjects more tangible and accessible to young learners, although critics of *Logo* claimed that it was too difficult for children (Pea, 1983; diSessa, 1997). Papert's notion of 'hard fun' grew, partly, in response to these critiques. Moreover, much of the early research surrounding the evolution of constructionism (e.g. Harel, 1991; Harel and Papert, 1991a; Kafai, 1995) found that using *Logo* to complete creative design tasks such as programming computer games was inherently difficult because it involves an integration of skills across a wide range of subjects, as well as meta-cognitive skills such as planning, reflecting and self-organisation (Kafai, 1995). Such complex projects require learners to develop strategies for dealing with complexity (Kafai, 1996), rather than avoiding it, and this is recognised as an important element of learning to learn in constructionist ideology.

4. Learning to learn

"Many students get the idea that 'the only way to learn is by being taught'. This is what makes them fail in school and in life. Nobody can teach you everything you need to know. You have to take charge of your own learning" (Papert, 1999a).

Much of the early research surrounding the theory of constructionism is concerned with the design of learning environments which enable children to learn how to learn. In

developing *Logo,* Papert designed a 'microworld' that he hoped would give children access to new ways of learning and thinking; as they encountered basic programming concepts for the first time and learned to communicate with the computer they were also engaged in a process of learning how to learn.

Constructionist learning environments

Papert was interested in how the design of computer-based learning environments could support children's cognitive development (Papert, 1980b: 161). He designed the *Logo* turtle as a 'transitional object', so called because it exists in the child's world but enables the child to make contact with abstract mathematical ideas, such as distance and angles, since these have to be defined to make the turtle move. This idea of 'transition' is further enabled in constructionist learning environments since they provide a context for children to develop their learning skills by allowing them to construct 'transitional theories' as they explore and build and play. For Papert, transitional theories are part of the process of learning to think (Papert, 1980b: 132), not deficiencies or cognitive gaps in understanding but ways of "flexing cognitive muscles, of developing … the necessary skills for more orthodox theorizing" (Papert, 1980b: 133).

Papert hoped that *Logo* would remove much of the complexity of learning, since it allowed children to play freely with its elements. As they interact and explore with the turtle, they develop personal understandings based on their previous experiences (Papert, 1980b: 162). They use the turtle as an 'object to think with' and this makes learning more visible to them, since the turtle provides immediate visual feedback of their ideas. The turtle supports learning also because it enables 'syntonic' learning, where children can identify 'bodily' with the learning material. As children manipulate on screen, visual objects or computer-controlled robots, for example, they can use their bodies to try out angles, distances, rotations (Papert, 1980b). This way of learning establishes a connection between the child and the learning material, and is qualitatively different from what Papert refers to as 'dissociated' learning (Papert, 1980b: 47), where there is no connection between the child, the learning material, or the outcome. In this way, constructionist learning environments such as *Logo* humanise learning "by permitting more personal, less alienating relationships with knowledge" (Papert, 1980b: 177). They also give rise to creative exploration, since children are able to take charge of the turtle and use it for their own purposes, and this leads to invention, and creativity (Papert, 1993: 176), outcomes which are less likely to arise in instructionist uses of the computer.

In addition, working within the visual plane of turtle graphics allows access to new modes of thinking (Papert, 1980b: 98) and expression. For example, *Logo* enables children to do new things with words; they learn that they can now draw a circle or make an object move, or repeat an action with words. They learn to 'think dynamically', to control things which move (Papert, 1980b: 94). They learn to create structures in a modular fashion, which helps them to build and to debug their programs. This aids learning because "When knowledge can be broken up into 'mind-size bites' it is more communicable, more assimilable, more simply constructible" (Papert, 1980b: 171).

In these ways, Papert saw constructionism as a mode of learning more suited to some kinds of learner than 'instructionism'[4], which he argued was prevalent in US schools at the time. For Papert, the focus in education on improving teaching methods and curricula was misplaced. For significant change to occur learners needed to be given better opportunities to construct (Papert, 1990b: 3), and making available computers, appropriate software and constructionist activities was one way of doing this. Where traditional instructionist approaches to education define what children need to know, constructionism asserts that children will learn better by finding for themselves the specific knowledge they need, when they need it; computers and good learning activities support this endeavour (Papert, 1993: 139).

Styles of learning

Another feature of constructionist learning environments is that they support different styles of learning. Papert observes that children are natural learners and learn a "vast quantity of knowledge" without direct instruction before they go to school (Papert, 1980b: 7). This natural, spontaneous learning that occurs as part of living in the world he refers to as 'Piagetian learning'. To his mind, this form of learning should be given more status in schools, since "the best learning takes place when the learner takes charge", rather than when they are told what to do and what to learn (Papert, 1993: 24). Papert suggests that constructionist learning environments, such as *Logo,* foster Piagetian learning (Papert, 1980b: 187) because they enable children to learn through exploration, without direct instruction, give the learner control of their own intellectual activity and allow for personal expression.

But while Papert values the kind of learning which happens without deliberate teaching, he does not advocate eliminating instruction or leaving children to their own devices,

[4] Instructionism refers to "educational practices that are teacher-focused, skill-based, product-oriented, non-interactive and highly prescribed" (Jonassen, 1996).

rather the focus should be on "supporting children as they build their own intellectual structures with materials drawn from the surrounding culture", introducing new constructive elements into the culture of schools and making suitable learning material available (Papert, 1980b: 31-32).

In addition to his concerns with how computers can enhance the learning process, Papert was also interested in the different ways in which children approach their work (Papert, 1991b: 5). Constructionism values a 'bricoleur' approach, where children are "guided by the work as it proceeds rather than staying with a pre-established plan" (Papert, 1991b: 6). Papert refers to this style of learning as 'tinkering' or 'bricolage'[5]. This way of working is playful, exploratory and experimental and in Papert's view, is as valid as more formal, structured approaches (Papert, 1993: 144) and the "curriculum driven learning" characteristic of traditional schools (Papert, 1980b: 156).

Concrete learning

Extending this idea of different styles of learning, Papert argues that there is also a need to rethink what sorts of knowledge, and what ways of knowing, should have 'privileged status' in schools (Papert, 1993: 19). He considers that the value placed on abstract, formal knowledge in schools is discriminatory and oppressive to those who do not learn in this way (Papert, 1993: 148) and argues for an 'epistemological pluralism', which values concrete as well as abstract forms of knowledge (Turkle and Papert, 1990).

While Piaget's theory of cognitive development distinguishes between 'concrete' and 'formal' thinking and regards formal, abstract reasoning as more advanced (Piaget and Inhelder, 1969), for Papert, concrete thinking is no less valid (Turkle and Papert, 1990; Papert, 1993: 151) and he argues that schools need also to value more concrete ways of knowing (Papert, 1993: 137). The computer has a unique role to play here in that it can concretise (and personalise) formal or abstract concepts and in so doing support children in their development from child to adult thinking (Papert, 1980b: 21). For example, working with a computational entity such as *Logo's* turtle allows children to externalise, or to make concrete, their ideas (Papert, 1980b: 145) because they can be seen; once seen those ideas can be reflected upon, evaluated and amended if necessary. Furthermore, whilst providing concrete materials for learning, environments such as *Logo* allow children to create something concrete in turn. At the same time

[5] Papert borrows the term from Claude Levi Strauss, who in his book *Structural Anthropology* (Levi Strauss, 1963-76) uses the word 'bricolage' to refer to improvisational methods of theory building in primitive science.

they bring children closer to formal, abstract thinking because as they write and test computer programs they learn about *computational thinking*, the sort of thinking that is necessary when working with computers, such as procedural thinking, systematic thinking and problem solving, all forms of abstract, formal thinking (Papert, 1980b: 174).

Collaboration

But constructionism is not solely defined by computer-based materials and activities. The surrounding environment also plays an important part, in terms of the nature of the relationships (with knowledge, with others) which are set up within it. Constructionist learning environments are defined by their "interactionist and affective characteristics" and by their affinity with collaborative learning and the social construction of knowledge (Kafai and Harel, 1991: 85). Moreover, the traditional notion of collaboration as working directly with others, is extended by the idea of "collaboration through the air" (ibid.: 88), where learners interact with free-flowing ideas and concepts present within the learning environment of the community of practice of the classroom. The *integration* of these different ways of working collaboratively characterises the constructionist approach (Kafai and Harel, 1991: 103).

Papert suggests that another strategy to facilitate learning is to improve the connectivity within the learning environment, by developing learning cultures, rather than focussing on interventions with individuals (Papert, 1993: 105). Within such cultures children learn by sharing their designs and experiences with others (Kafai et al., 2009b: 81). It is a more collaborative, collegiate way of learning and importantly, learners of all abilities can make a contribution to it. Furthermore, constructionist learning activities, which focus on the production of shareable artefacts, designed for real purposes and audiences, also emphasise the social and the collaborative dimensions of the theory. Other forms of connectivity are strengthened in such open-ended projects because children must be able to access knowledge when they need it and this may involve collaborating with peers, and 'experts', and accessing wider networks via the internet (Papert, 1994).

5. Taking time

"The fifth big idea is taking time – the proper time for the job. Many students at school get used to being told every five minutes or every hour: do this, then do that, now do the next thing. If someone isn't telling them what to do they get bored. Life is not like that. To do anything important you have to learn to manage time for yourself. This is the hardest lesson for many of our students" (Papert, 1999a).

An important feature of constructionist philosophy is the idea that effective learning is more likely to occur when children are given time to become personally, intellectually and emotionally involved in their work (Papert, 1970: vii-4). However, this is unlikely to occur in the regular school day, where children are expected to switch in and out of several projects, in a fragmented timetable, conditions which do not allow for "personal appropriation and expression of personal intellectual style" (Harel and Papert, 1991b: 67). By contrast, longer projects enable learners to try several ideas, to have the experience of putting something of oneself in the final result (Papert, 1970: vii-8). Units of work which arise out of the constructionist approach offer learners extended routes for learning because they are "not done and dropped but continued for many weeks", allowing pupils "time to think, to dream, to gaze, to get a new idea and try it and drop it or persist, time to talk, to see other people's work and their reaction to yours" (Papert, 1991b: 4).

On a practical level, computer-based constructionist activities which involve the creation of digital artefacts demand extended time allocations since they are often design-based, open-ended tasks, which integrate multiple processes, skills and disciplines. Early constructionist research describes software design projects which took several months to complete (Harel, 1991; Kafai, 1995) and identifies this long period of involvement as crucial for learning the programming and design concepts required to develop software (Kafai, 1995: 14). More recent constructionist curricula also span extended time frames: *Globaloria* (Harel Caperton et al., 2010) courses demand 40-55 hours; *Gamestar Mechanic* (E-line Media, 2013) requires a semester long, 50 plus lessons.

For Papert, giving children time to learn something is 'an obvious principle' (Papert, 1993: 89). Learning takes time because the connections and associations which are part of it do not come all at once but emerge gradually, almost as an act of 'cultivation' (ibid.: 104). Moreover, immersion in an extended project gives children a sense of what it is like to carry out a complex project and to manage the problems which arise out of their own work and ideas (Papert, 1994).

6. Freedom to get things wrong

"You can't get it right without getting it wrong. Nothing important works the first time. The only way to get it right is to look carefully at what happened when it went wrong. To succeed you need the freedom to goof on the way" (Papert, 1999a).

Constructionism promotes the idea that it is important for learning to give pupils the freedom to get things wrong. Papert refers to this as the 'biggest idea of all' and suggests that, in the context of school, children often have a model of learning which is dominated by a sense of being 'right' or 'wrong', which inhibits learning and which is at odds with how many of us actually learn in the real world, by being 'vaguely right' or 'fumbling' toward a solution or understanding (Papert, 1993: 167). Computer-based constructionist activities offer an alternative model for learning where concepts of 'right' and 'wrong' are less applicable. Whilst many children "lack a model of understanding something through a process of additions, refinements, debugging and so on" (Papert, 1970: 9), learning to program introduces them to an alternative epistemology, where isolating and correcting errors, evaluating and refining procedures and making things work is an iterative process. Errors in this context are to be studied, not avoided (Papert, 1980b: 61), since they are a source of information (Papert, 1993: 184).

Moreover, in programming activities 'right' and 'wrong' are not absolutes – but exist on a continuum - a program might have a 'bug' but still function to some extent. And these bugs can be fixed by exploration and play (Papert, 1980b: 62). Such constructionist approaches change children's relationship with 'right' and 'wrong' and this is important for their development as learners because they see that problems can be solved in stages, that mistakes do not invalidate the whole enterprise.

These changing perspectives of right and wrong are joined by other shifts. For Papert, partial and qualitative knowledge constitutes 'good' knowledge, as much as complete and quantitative forms (Papert, 1993: 21). Learners can begin by "knowing something in a very fumbly sort of way" (Papert, 1993: 64). In any case, he acknowledges, most things are only partially understood (Papert, 1980b: 117) and as developing learners, children can benefit from coming to realise that partial understandings are inevitable when dealing with understanding complex ideas.

In traditional epistemology, knowledge is valued for being correct and considered inferior if it lacks precision (Papert, 1993: 185), but alternative epistemologies are introduced when dealing with constructionist programming activities. As an example Papert refers to the difference between the precise commands needed to create programs to draw turtle graphics and the 'vague' and general programs which can be written to control a turtle equipped with sensors which makes use of feedback (Papert, 1993: 187). The key idea here is that in engaging with different sorts of programs (drawing a circle in *Logo*, programming multiple agents in simulations, writing programs

which make use of feedback) children are acquainted with practices which are less concerned with precision and correctness and more to do with emergent thinking. When engaged in programming, 'knowing the right answer' is less important, since what matters is getting things to work (Fonseca et al., 1999), and children learn that there may be multiple solutions to the problems they encounter.

7. Teacher as co-learner

"Do unto ourselves what we do unto our students. We are learning all the time. We have a lot of experience of other similar projects but each one is different. We do not have a preconceived idea of exactly how this will work out. We enjoy what we are doing but we expect it to be hard. We expect to take the time we need to get this right. Every difficulty we run into is an opportunity to learn. The best lesson we can give our students is to let them see us struggle to learn" (Papert, 1999a).

In constructionist learning environments the teacher is present as a co-learner and the mode of learning is less dominated by lesson plans or a set curriculum (Papert, 1980b; Kafai, 2006a). Pupils are given ownership of their learning and encouraged to manage tasks and timing themselves. Instruction itself is more distributed and negotiated, "constructed in interactions between the teacher and students" (Kafai, 2006a: 36). In this vision of learning experts and novices learn together for real purposes (Papert, 1980b: 179), activities are participatory and authentic. Teacher interventions are driven by pupils' experiences and interests and they share their own learning discoveries and responses to activities. Importantly, the flow of ideas and instruction is reciprocal.

Papert's idea of teacher as co-learner was partly pragmatic: when *Logo* was first introduced in schools in the 1980s, the use of computers and of *Logo* itself was a new venture for many teachers. Children saw their teachers learning through exploration and from mistakes they had made. While other collaborations were 'fictions' where the teacher already knew the answers, in *Logo* projects, teachers new to programming learned alongside their pupils and thus were engaged in real intellectual collaboration (Papert, 1980: 115). Moreover, writing open-ended programs throws up authentic, individual problems that neither teacher nor learner will have encountered before. In sharing these problems and the experience of finding solutions to them, children "participate with a good learner in an act of learning" and learn to become good learners themselves (Papert, 1999b: ix). From a constructionist perspective, teachers should do a lot of learning in the presence of children and in collaboration with them (ibid.: xv) since this encourages children to view learning as a lifelong process.

Beyond school settings, constructionism promotes the idea of 'teacher as co-learner' in after-school clubs, where the interaction between individuals is more akin to an apprenticeship model (Kafai et al., 2009b). In Computer Clubhouses for example, 'teachers' are mentors, volunteers from business or undergraduate students, who learn about new creative technologies alongside club members (Kafai et al., 2009b). Mentoring in these contexts is conceived as a form of partnership, where both parties have something to offer the other.

8. Using computers to learn in a digital world

"We are entering a digital world where knowing about digital technology is as important as reading and writing. So learning about computers is essential for our students' futures but the most important purpose is using them now to learn about everything else" (Papert, 1999a).

Constructionist learning activities bring to the fore areas of knowledge which are crucial in the modern world if young people are to participate with understanding in the *construction* of what is new (Papert, 1999b: ix). Papert refers to this area of knowledge as 'cybernetics' (Papert, 1993), a subject which incorporates computational thinking, systems thinking and programming. In order for this knowledge to be accessible to young learners and to make it so, a different culture for computer use needs to be developed. In this context, Papert presents constructionism as a conceptual framework for how computers could be used in education, so that "computational material [is used] as an expressive medium" (Papert, 1991b: 4) and a 'medium for thinking' (ABC Online, 2004) rather than a new tool added to old practices (Papert, 1997).

However, whilst "computers … provide an especially wide range of excellent contexts for constructionist learning" (Papert, 1991b: 8), computers themselves are not the primary concern. Papert resists 'technocentric' views which ascribe more importance than is appropriate to technology alone as an agent of change in education (Papert, 1990a). Rather, he is interested in how computers can be harnessed to positively affect the nature of the learning process and the production of knowledge by students.

While Papert lamented school as a paper-based system unsuitable for digital society (ABC Online, 2004), he acknowledged that widespread use of educational technology would inevitably lead to new ways of thinking and learning. This he saw as an evolutionary process, which would be augmented by the use of computers in the home (Papert, 1996b), since it takes time for technology to give rise to new practices and the new cultures that support them.

The development of *Logo* and its application in schools was itself an evolutionary process. Early *Logo* projects were short-term tasks commonly used to introduce pupils to programming and turtle geometry (Papert, 1997). The later development of *Lego/Logo* (Sargent et al., 1996) allowed pupils to construct robots and machines which were controlled by *Logo* programs. This gave pupils access to new types of programming structures, which involved the use of feedback given by sensors. Short-term projects gave way to extended projects in which children used new versions of *Logo* (*LogoWriter, MicroWorlds*) to design and produce real products such as computer games (Harel, 1991; Kafai, 1995), where programming creates on screen action arising from the interactions between objects and the properties of objects. In such ventures, computers are used for 'real world' purposes (Papert and Solomon, 1971), rooted in children's culture and learning is more authentic because much of it arises out of the need to achieve particular effects in context (Papert, 1994).

To Papert's mind, computers "should serve children as instruments to work with and to think with, as the means to carry out projects, the source of concepts to think new ideas" (Papert, 1993: 168). His concern was that computers were being used in limited ways, such as for computer aided instruction, which neither harnessed the computer's potential, nor improved children's learning experience. In contrast, by learning to program computers children learned about how computers work and this was important at a time when such technology was becoming increasingly opaque to lay people (Papert, 1993). Papert wanted to steal programming from the technologically privileged and give it to children (ibid.: 180) and to provide routes to 'softer', more playful relationships between children and technology, such as those which are set up when children create programmable entities that they are interested in or make computer games, for example.

In Papert's view the focus on the presence of computer technology in schools, on the development of mechanical skills or the use of specific 'office' software applications, or on how such technologies give access to information, denies children any deep understanding of computing or agency over the technology. Papert wanted to see computers used as "something the child himself will learn to manipulate ... thereby gaining a greater and more articulate mastery of the world, a sense of the power of applied knowledge and a self-confidently realistic image of himself as an intellectual agent" (Papert, 1970: vii-1).

3.3 Summary

This chapter has outlined the learning theory of constructionism, which evolved alongside the introduction of computers in schools, and has presented Papert's own synthesis of its main principles as 'eight big ideas'. In summary, the approach argues that children learn best by doing, and specifically, by making computer-based artefacts which have personal and cultural meaning for them. In making such artefacts, young people are engaged in a process of constructing their own understanding of the knowledge required to make it. Part of that process involves learning how to find the knowledge they need and how to solve problems and correct mistakes along the way. This is best done when children are given sufficient time to become personally involved in what they are learning and making. Important also is to give children some control over their learning and some freedom over how they approach their work. Acknowledging that making things with computers often involves some kind of programming and that such activities can be difficult, the theory seeks to find ways to support learners in their endeavours. These include situating learning in the context of use, encouraging collaboration between teachers and peers, and making available suitable computer-based learning environments which provide concrete opportunities for learning.

From being a theory of learning 'in evolution' (Papert, 1991b), constructionism is nowadays a commonly cited theoretical framework within educational research (Bulfin et al., 2013). In particular, this theory of learning informs much of the research into computer game authoring from a programming perspective (see Harel, 1991; Harel and Papert, 1991b; Kafai, 1995; Kafai and Resnick, 1996a; Kafai et al., 2009b) and thus provides an appropriate theoretical frame for the current study.

3.3.1 Constructionism and the current study

The current research explores constructionism as a suitable approach for learning how to design and program a computer game, using *Game Maker*, in the context of the mainstream UK secondary ICT curriculum. This focus complements and extends the bulk of constructionist research which has been conducted in the United States, using *Logo* to teach primary aged pupils about mathematics (Harel, 1991; Kafai, 1995; Kafai, 2006b) and more recently, using *Scratch* in out-of-school contexts to create multimedia artefacts (Kafai et al., 2009b) and in the primary phase to teach about programming and science (e.g. Baytak and Land, 2011a, 2011b).

In particular this study embraces the notion that pupils should be producers as well as consumers of digital media and gives them an opportunity to use computers as a means of creative expression to make a product of personal, social and cultural relevance to them, an enterprise not strongly present in the National Curriculum programme of study in operation until 2012. It explores pupils' perceptions of the constructionist approach they followed and their responses to the outcomes they produced and considers to what extent such ways of working are effective in learning basic programming concepts and practices.

It explores whether constructionism is a suitable pedagogic approach for introducing new curricula (a new computing curriculum becomes statutory in September 2014), given that the constructionist imperatives of teacher as co-learner and learning by doing are likely to become practical necessities for those teachers who have little experience of programming or of teaching textual programming languages. In such a climate the importance of pupils themselves learning how to learn is brought to the fore and so this study is also concerned with whether extended, open-ended projects and learner-directed and collaborative working patterns are successful strategies in contemporary contexts.

Chapter 4 Research design and methodology

4.1 Introduction

This chapter outlines the research design and discusses the methods that were selected to collect and analyse the data. It additionally describes how the research design and methods chosen are appropriate to the research questions and purpose.

The rationale for selecting a qualitative approach is presented. Within this broader framing, the use of case study as the overarching design is justified. The data collection methods are described and issues regarding the validity and reliability of the data are considered. The data analysis strategy used and the ethical procedures followed are also outlined.

4.2 Qualitative research

The purpose of the research is to explore pupils' perceptions about the process and outcomes of their learning during the game authoring activity, and following on from this, to gain an appreciation of what they learned, what they valued and what difficulties they encountered. The research strives for depth of understanding in these areas, and it is therefore a qualitative enquiry. A qualitative approach was selected as the most appropriate means of addressing the particular research questions posed in this study, since the research activity was classroom-based and the particular unit of work pupils followed was essentially creative and ongoing. The research design was therefore concerned to capture aspects of the experience of Year 9 pupils as they learned to create computer games for the first time. However, whilst the study has taken a qualitative focus, some quantitative data are presented as indicators of extent of certain findings within the group.

4.2.1 Qualitative research and computing education

In recent years qualitative research methods have become more common in computing education research (Kinnunen and Simon, 2012) and such approaches have much to offer in the field (Berglund et al., 2006). However, since quantitative approaches have predominated in research into the teaching and learning of programming (Sheard et al., 2009) there is a need for more 'pedagogically anchored qualitative research' which

makes a practical contribution, based on established theoretical frameworks, in how to teach and learn aspects of computing (Berglund et al., 2006; Sheard et al., 2009). This is particularly important in terms of the current situation in the UK, where teachers are preparing themselves to deliver a new Computing programme of study from September 2014 and discussions of the pedagogy of programming and computing are beginning to surface in online forums and blogs (e.g. CAS, 2008a; Guzdial, 2009; ScratchEd, 2009; TES, 2013). The current study adopts a qualitative approach and takes as its theoretical frame the learning theory of constructionism (see Chapter 3), applying it to computer game authoring as a context for learning basic programming concepts. In so doing it contributes to the field of qualitative research in secondary computing education.

4.3 Rationale for selecting case study

A case study is a detailed description and analysis of a bounded system (the case) (Merriam, 2009: 40) and investigates an area of interest within its real-life context (Yin, 2009: 18), collecting its data from multiple sources of information (Creswell, 2007: 73). It allows the researcher to conduct an "intensive, holistic description and analysis of a single instance" (Merriam, 1998: 21). Case study was selected since it allows the study of an evolving situation, that is, the introduction of a unit of work in game authoring in the Key Stage 3 ICT curriculum. The case is 'intrinsically interesting' to the researcher and is being studied to achieve as full an understanding of the phenomenon (game authoring) as possible (Merriam, 2009: 42). It is therefore an exploratory case study.

In this research, the case is a group of Year 9 pupils who completed a unit of work in computer game authoring over an eight week (16 x 50 minute lessons) period. The unit of analysis is both the group and the unit of work, which constitute a bounded system (Merriam, 2009: 41). A single case design was chosen on the basis that the class selected is a 'typical' case of a wider population of Year 9 pupils. Lessons learned from typical cases are assumed to be informative about the experiences of the average [child/class] (Yin, 2009: 48).

Within the case study, several methods of data collection were selected to strengthen the internal validity of the data: pupil paired learning conversations; constructed computer games and other pupil documents; group interviews and artefact-based pair interviews. According to Yin (Yin, 2009: 11), this ability to deal with a variety of

evidence is one of the method's strengths.

Cases studies are useful in presenting information about areas of education where little research has been conducted (Merriam, 1998; Gerring, 2007), because they can provide a base or a starting point for further investigation. In the current study one group of Year 9 pupils learned about game authoring using a particular piece of software, which has its own pedagogy, and thus makes a context- and tool-bound contribution to the field. Its findings may be usefully extended by further studies which investigate other tools, other populations or other theoretical frameworks. As outlined in Chapter 2, few contemporary studies of computer game authoring in the UK secondary ICT curriculum exist, so the present case study offers insights, experiences, and perspectives which build up the field's knowledge base and may help to structure future research (Merriam, 2009: 51). In this regard, the case study method can "suggest to the reader what to do or what not to do in a similar situation" (Olson in Merriam, 1998: 30) and this can be fruitful when considering new practices, such as the introduction of a new unit of work, area of learning or a new approach, as in the case reported here.

Key constructionist research (Harel, 1991; Papert, 1993; Kafai, 1995) also incorporates case study within its design, focussing on the use of particular learning environments (software) and implementations of game authoring and programming curricula, as in the current study. Case study is also commonly used in broader computing education research when specific courses or tools are presented (Berglund et al., 2006).

4.3.1 Limitations of case studies

But the particular features of case study which provide the rationale for its selection also present limitations (Merriam, 2009: 51). "Case studies can oversimplify or exaggerate a situation" (Guba and Lincoln in Merriam, 1998: 42) or claim to present the 'whole picture', whereas in fact they are only a part of it. Case studies are also limited by the sensitivity and integrity of the researcher. Ethics too can be compromised in case studies, where, "an unethical case writer could so select from available data that anything could be illustrated" (Guba and Lincoln in Merriam, 1998).

The case study method is otherwise criticised for lacking reliability, validity and generalisability, but these are not the chief concern of qualitative research (Merriam, 1998). Rather, the focus is on understanding the particular case (Evers and Wu, 2007: 201) and since a case study does not represent a 'sample', it does not need to be generalisable to wider populations.

To strengthen the reliability of a case study in the face of such criticisms, Yin recommends the development of a case study database (Yin, 2009: 45) to store data and procedures followed, so that the research could be replicated. In terms of the current study, a database of pupil voice recordings and interview data, transcripts, interview schedules, and the coding system used at the analysis stage was created and stored in *NVivo 8* (QSR International, 2008). Additionally, documented research procedures, data collection guidelines and a scheme of work were produced, which serve to strengthen the reliability of the research.

4.4 Research design

4.4.1 Pilot study

A pilot study was completed with one group of Year 9 pupils (n=23) in the autumn term (2009) preceding the main study, during which the research instruments were tested and the scheme of work was trialled. Pupils worked in pairs to create computer games using *Game Maker*. Pupils' journal entries were transcribed and coded. From field notes recorded throughout the pilot study, the following issues emerged; these were addressed in the main study by making the changes indicated in italics:

- Pupils had problems creating a coherent original narrative for a game. *Greater emphasis was placed on the narrative aspects of game authoring in the scheme of work.*

- Although pupils were excited by the idea that they are being recorded as they worked, there was an initial self-consciousness in using the voice recorders. This 'inhibiting' effect is commonly observed in classroom based research, where pupils are being observed or recorded (Edwards and Westgate, 1994). It was likely that as pupils became accustomed to recording their experiences in this way they would find it less intrusive. *With regard to the main study, to minimise the effects of this self-consciousness, data collected in the early sessions was not included in the analysis.*

- Unspoken thoughts, feelings and impressions could not be captured by the digital recordings, so pupils were asked to refer to these in a journal, set as a homework task. The disadvantage of this method was that these journal entries were removed from the immediate instance and so may not offer a reliable

record. Pupils were resistant to using a journal to record their work in progress and entries were mainly descriptive rather than reflective or analytical. *For the main study, pupils were asked to make their journal entries on the same day that they had their ICT lessons, so that they would be able to remember their experiences more accurately. Prompts were given for what to write in the journals and these asked for information that did not rely solely on memory, but instead gave pupils an opportunity to express their opinions, experiences, and the difficulties they encountered.*

- During the pilot, it became apparent that the cognitive load (Sweller, 1994) of learning new concepts, new software, new vocabulary and new activities (game authoring) was a challenge for many pupils. *For the main study more resources were provided to support pupils in the game authoring process, and the scheme of work was modified to deliver targeted sessions relating to those concepts that pupils found difficult in the pilot study.*

4.4.2 Selecting a sample

A purposive non-probability sample (a non-random group selected for a specific purpose - to trial the game authoring activity) on the basis of their typicality (of year 9 pupils), was selected, "in the full knowledge that it does not represent the wider population; it simply represents itself" or instances of itself in a similar population (Cohen et al., 2007: 113). A purposive non-probability sample is often used in small scale, case study research (ibid.), since it is not the intention of such studies to generalise findings to a wider population.

The criteria for selecting the pupils in this case study were that:

i) They should reflect the spread of ability in a typical mixed ability Year 9 group.
ii) There should be an equal mix in terms of gender.
iii) They should be timetabled 2 x 50 minute lessons of ICT per week.

In Creswell's terms, the sample was achieved by selecting an 'accessible', 'ordinary', 'typical' case (Creswell, 2007). Purposive sampling was achieved within the case in terms of which pupils were selected as members of the interview groups, and for the paired interviews. Three boys and three girls were selected for each group interview, and of these, two were selected from each of the higher, average and lower ability ranges. For the paired interviews, four boys and three girls were selected to represent

a similar ability spread. Seventeen of the twenty-two pupils in the class were interviewed either as part of a pair or a group. Pupil voice recordings, authored games and documents were not sampled and all units produced were included in the analysis.

4.4.3 Participants

The participants were 22 Year 9 pupils (12 boys; 10 girls; 13-14 years old). In planning and developing their games they worked in self-selected pairs, apart from two pupils (one boy and one girl) who worked alone, by choice. One pair was mixed gender; the other 9 pairs were the same gender. Pupil journal entries and storyboards were completed on an individual basis.

The research was conducted in a high achieving school in South East England. Ten of the twenty-two pupils in the group achieved 'above average' values in their average Cognitive Abilities Test (CAT) scores; 7/22 of pupils achieved a CAT score of 120 or higher in one or more CAT measures, which suggests that the group was of above average ability with respect to national profiles. Ability was added as an attribute to each case (pupil) in *NVivo*, using Jesson band level, the performance indicator used by the research school (see OFSTED, 2008) as shown in Figure 8.

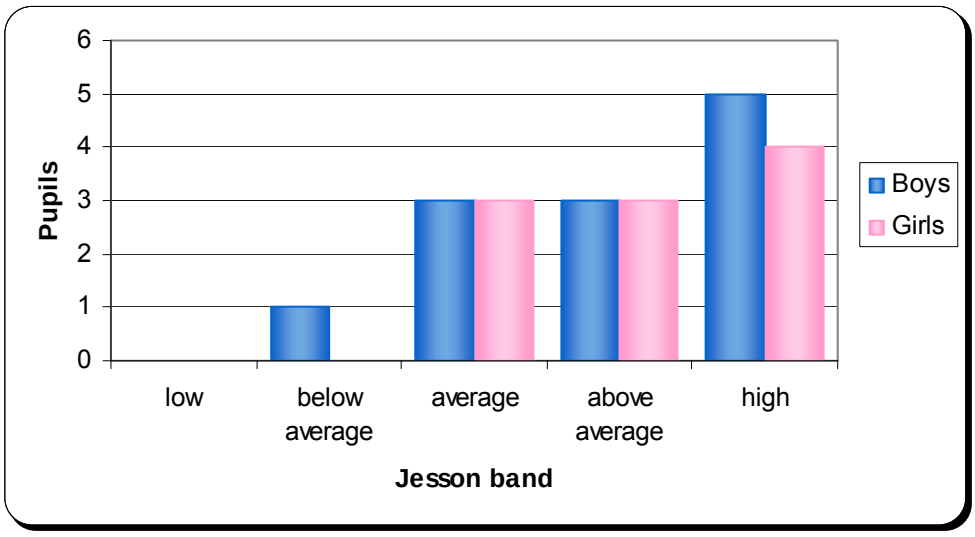


Figure 8: Pupils' Jesson band ability level

4.4.4 The scheme of work

This was the first time pupils had made a computer game as part of formal ICT lessons. Previous learning in visual programming was delivered in Years 7-9 using *Flowol 3* (Bowker, 2005) and used a flow chart paradigm to introduce pupils to

programming concepts including inputs, outputs, loops, decisions, sub-routines and delays.

In constructionist learning activities, children work on extended projects, learning by doing and finding for themselves the specific knowledge they need (Papert, 1993: 139), supported by teacher guidance and appropriate resources. Accordingly, the scheme of work spanned 16, 50-minute lessons and was structured to provide a frame and focus for each lesson, a mix of teacher-led, independent and pair work, a range of video, print and computer-based resources, and an integration of written, oral and computer-based activities (see Appendix 7). Homework was set once per week and asked pupils to write about their work in progress and to describe any problems or difficulties they experienced.

4.4.5 Working in pairs

Pupils were invited to work in self-selected pairs to plan and construct their computer games, as part of creating a collaborative, constructionist learning environment. In this respect, working in pairs can promote learner autonomy, since pupils are able to use the interactions with their partner and the software as potential sources of support, reducing their reliance on the teacher. Although a scheme of work provided a framework for the activity, pupil pairs were able to negotiate their own priorities on a lesson-by-lesson basis. In addition, pupils expressed a preference for working in pairs and had not been given a formal opportunity to work in this way on an extended project before.

Pair work as a teaching strategy brought several benefits. It allowed partners to share ideas and complete tasks collaboratively, an important characteristic of constructionist learning theory (see Chapter 3). Working in pairs may also have promoted and sustained pupil engagement, important for the completion of such a novel, complex, open-ended, activity. In their working conversations pairs were able to construct understandings between themselves and provided a source of intellectual and motivational support for each other (Vygotsky, 1978). Moreover, peer explanations may be better matched to pupils' existing understandings (Lewis, 2011) than other resources. On a practical level, pupils may feel more involved and be more actively engaged when working in pairs, rather than larger groupings. They are also more likely to succeed in cognitive tasks when they work in pairs (Kutnick et al., 2005: 47). This aspect of the research design also enabled the researcher to collect voice data of pupils' working talk as they co-constructed their games.

4.5 Data collection

Data were collected over an 8 week period. Several methods were used to capture multiple sources of data, to strengthen its internal validity.

4.5.1 Data set

The data set for this study consists of:

i) Ten transcripts of digital voice recordings of pupil pairs' working conversations (4 hours, 28 minutes).
ii) Two transcripts of group interviews. At the end of the project, semi-structured interviews were recorded with two groups of 6 pupils (3 boys; 3 girls), where they talked about their game authoring experience with each other (2 x 43 minutes).
iii) Three transcripts of artefact-based paired interviews, in which pupils' games were loaded and used as the focus (1 x 39 minutes, 1 x 33 minutes, 1 x 53 minutes).
iv) Twelve authored games.
v) Eighty-five pupil documents. Pupils documented their reflections on aspects of their work in an ongoing written journal and completed planning documents (storyboard, game design document, game interactions).
vii) Observation notes were recorded throughout the field work.

These multiple sources of evidence allowed for aspects of pupils' experience of the game authoring activity to be articulated from different angles. Borrowing Yin's diagram (Yin, 2009: 117), Figure 9 below illustrates the sources of evidence used to collect data in the current research.

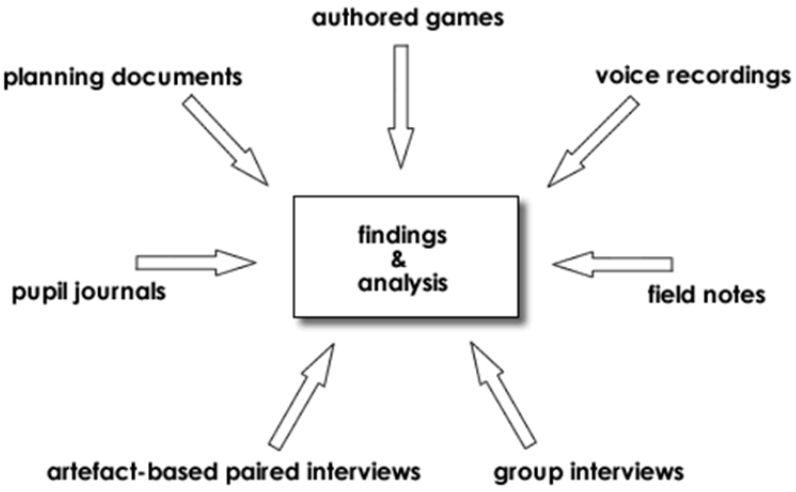


Figure 9: Convergence of multiple sources of evidence

The following section considers the strengths, assumptions and problems present in interviewing as a data collection method generally and of interviewing pupils in particular.

4.5.2 Interviews

Interview is an important source of information for case studies (Merriam, 2009; Yin, 2009) and was selected as a method of data collection since the research was concerned with exploring pupils' experiences and perceptions of the game authoring activity, and interview allowed pupils to express themselves in their own terms in this regard. The 'shared, negotiated' nature of interviews (Cohen et al., 2007: 151) allows for data to be constructed between researcher and pupils and was a useful tool in this classroom-based research.

Interview schedule

A semi-structured interview schedule was designed to achieve a measure of consistency between interviews (see Appendix 3). The schedule included a mix of more and less structured questions since these can be used flexibly, i.e. wording and question order can be varied (Merriam, 2009), whilst allowing for comparison between respondents and making data collection more systematic and comprehensive (Cohen et al., 2007). This format was selected to enable the researcher to be responsive to the groups and to individuals' responses, and to probe for deeper answers whilst maintaining an informal structure.

Questions included introductory questions to ease pupils into the interview, followed by key questions, framed around the research questions and open questions to enable respondents to talk about their experiences and understandings. Pupils were assured that there were no right or wrong answers and that all contributions were valued. Similar questions were asked in both the group and artefact-based pair interviews, however there were some changes in wording, and in the follow-up questions to the answers obtained.

The schedule included a mix of 'experience' and 'opinion' questions (Patton, 2002 in Merriam, 2009: 96) and "hypothetical, devil's advocate, and interpretive questions" (Strauss et al., 1981 in Merriam, 2009: 97). These different question types were selected to encourage pupils to share aspects of their experiences of the game authoring activity, although it is acknowledged that pupils' answers do not provide *evidence* of their experience, but only an account or representation of an experience

(Silverman, 2011: 181).

Interview sample

One of the strengths of the interview sample was that 17/22 pupils were represented in either the group or the artefact-based pair interviews. Group interviews were conducted with two groups of six pupils (3 boys; 3 girls) to ensure that both genders were equally represented. Pupils were also selected to achieve a balance in terms of ability. Groups consisted of two pairs (mm/ff) and two individuals (m/f). Working pairs chosen for the artefact-based interviews similarly represented the gender and ability mix of the class. Paired interviews were same sex (2 x boy pairs; 1 x girl trio), since 9/10 pairs elected to work with a partner of the same gender. One pair interview included a third pupil, to give one of the two pupils who worked alone the opportunity to contribute to the spoken data collected.

Group interviews

The group interview was selected as a data collection method since it promotes interaction and allows discussions to develop, increasing the possibility of a wide range of responses. Additionally, pupils may feel more comfortable being interviewed in a group and can support, prompt, influence, agree or disagree with each other leading to a more complete record (Cohen et al., 2007). The interviews were conducted in the same room in which pupils had their ICT lessons, during the teacher/researcher's non-contact time; pupils were withdrawn from other lessons to take part in the interviews. The interviews were recorded digitally and each spanned 43 minutes.

The data collected were socially constructed within the interaction of the group (Merriam, 2009) but will have been constrained to some extent by the presence of the teacher/researcher and the digital voice recorders. Although the interviews were informal, the teacher/researcher managed the transitions from one question to the next and encouraged all pupils to participate. Within each question pupils directed their own dialogue and the transition from one turn to another was managed cooperatively by pupils themselves.

Artefact-based paired interviews

Artefact-based interviews are useful to gain an understanding of pupils' experiences of creating digital media and give some account of the concepts they have used and the process they followed, as well as the product generated (Brennan and Resnick, 2012). They can also be useful insofar as they create a natural context for exploring ideas,

artefacts can trigger ideas that pupils might not articulate without them and they allow the interviewer to use pupils' natural language anchored by the specifics in the artefacts (Henderson et al., 2007).

Artefact-based interviews were conducted with three pairs towards the end of the project. In these interviews, pupils were asked to load their constructed games and to refer to them as appropriate. This added context, focus and immediacy to the questions and responses which occurred, whilst also reducing problems with retrospective bias and memory accuracy which may have weakened the reliability of the group interview responses. Using the games within the interviews was supportive to pupils because it gave them a visual point of reference for their responses and something tangible to talk about.

Limitations of the method are that it is time consuming - and although four games were discussed (three pair and one individual) this only represents 1/3 of the games. Even though the presence of the artefact may have reduced problems with memory accuracy, because the games were created over time and the interviews were held at the end of the unit of work, pupils' ability to remember details remains an issue.

Validity and reliability of interview data

The main concern with interview data is the extent to which the questions asked collect reliable and valid information. The interview questions were based on the research questions, to increase the likelihood that pupils' responses would yield data relevant to the enquiry. To minimise the potential for misunderstanding what respondents said, or respondents not understanding what was being asked, the researcher checked her interpretations with pupils by asking follow-up questions or by probing to achieve greater clarification. She also endeavoured to phrase questions clearly and simply and to maintain an informal, conversational and naturalistic tone throughout the interview.

However, in this classroom-based research, the validity and reliability of the interview data is challenged by the 'asymmetries of power' (Cohen et al., 2007: 152) between interviewer and interviewees, which in this case existed on several levels: the researcher was the data collector, an adult, and the teacher. Whilst a power differential is acknowledged, the researcher aimed to reduce its effects in the following ways: in the group interviews the presence of five other pupils may have made the interviews less intimidating (Cohen et al., 2007: 374); the power differential may also have been reduced because the researcher had been the interviewees' ICT teacher for almost

three terms, so there was a pre-existing, positive working relationship between both parties. Some commentators (e.g. Gosling, 2007) suggest that less of a power differential now exists between pupils and teachers because of the rise of so called 'student-centred' approaches. Indeed, the constructionist philosophy on which the study was based was conceived to empower young people, not only in terms of their relationship with technology, but also as learners. In this regard relations in the classroom are more negotiated and collaborative. The researcher sought to create the same feeling in the interviews by making it clear in the interview preamble that there were no right or wrong answers and that pupils were not expected to answer questions directly to the interviewer, but could use the questions as prompts for a discussion between themselves. Interviews were conducted in the group's 'normal' ICT classroom, although some pupils had been withdrawn from other lessons to be interviewed, so the event was 'unusual' in that respect and may have affected their responses. At the same time the interviewees themselves had power, in so far as they could choose to contribute or not, pay more or less attention to the questions asked and answer more or less sincerely (Cohen et al., 2007). In practice, pupils seemed to value being interviewed and wanted to share their views with the researcher and with each other.

A power differential may also have existed in terms of how much both parties knew about the interview topic. Certainly among the pupils themselves there were different levels of 'expertise' and those who felt that they were less 'knowledgeable' than others in the group may have been less forthcoming in answering some questions. To remedy this, the researcher endeavoured to draw all pupils into the conversation and to ask a mix of knowledge, opinion and experience questions (Merriam, 2009: 96). As a teacher, the researcher had more experience of using the software, and knew more about the process of game authoring it involved, but in other respects was a 'co-learner' and did not have the same experience of playing computer games as the pupils. Additionally, as part of an exploratory case study the questions were designed to investigate pupils' opinions, perceptions and experiences, so in this respect the teacher/researcher was not more knowledgeable than the pupils.

Tensions may also have existed in terms of the relationships between pupils in the interview groups and this may have impacted on the responses received. Some pupils may have felt intimidated by others in the group, or by certain questions, others may not have wanted to disagree with friends or recognised 'experts'. Some pupils may have given what they perceived to be socially desirable responses or answers they thought the interviewer as teacher might want to hear (Cohen et al., 2007). Some

questions involved asking pupils what aspects of game authoring they found difficult, a potentially sensitive issue, in so far as some pupils may not have felt comfortable acknowledging that they found aspects of the activity difficult, since to do so may be to admit a perceived lack of understanding or 'intelligence'.

Another concern with the reliability of the interview data is that it was subject to pupils' memory accuracy. Since the interviews were conducted at the end of the game authoring activity, some of the questions asked may have been difficult for pupils to answer because they may have referred to items which pupils could not remember accurately. Retrospective bias may also have reduced the reliability of interview responses.

The interview was also time limited and this will have had an impact on the amount of data collected. However, it was important to limit the duration of the interviews in terms of pupils' stamina for answering questions in an interview situation; the optimum time was about 45 minutes - and 'interviewee fatigue' (Cohen et al., 2007: 349) may have affected the reliability of the data collected towards the end of the interviews.

Finally, the interview data collected were 'uneven' in terms of who contributed. Some pupils talked more than others - this may have been because they had more to say or because they were more confident about expressing their views in a comparatively formal setting. The 'unspoken' views of other pupils is 'lost' data in terms of the interview transcripts - but these pupils' experiences of the game authoring process also find expression in the journal entries they wrote, the games they created and the digital voice recordings they contributed to.

Given all these factors, it is acknowledged that the data collected in the interviews is an incomplete record of what pupils thought, knew or experienced, but in so far as the interview data was triangulated with other forms of data, its validity lies in the extent to which emerging themes are corroborated by those.

4.5.3 Digital voice recordings

The purpose of asking pupils to record their working conversations was to enable the researcher to gather data about work in progress. Allowing pupils to manage the recording of their talk enabled them to decide what they considered to be important and to convey this in their own terms. To support pupils in 'knowing what to say' if they 'got stuck for words', a prompt sheet (see Appendix 4) was distributed which listed topics

they could talk about until they felt confident enough to use their own ideas. Although this prompt sheet may have directed pupil talk in the early stages of the data collection, pupils later discarded the sheets as they became more accustomed to talking about their work and more immersed in creating their games.

Although some pupils may have found the presence of the voice recorders intrusive or inhibiting (Edwards and Westgate, 1994), it was envisaged that this inhibition would decrease as pupils became habituated to using them. Thus several lessons were recorded at the beginning of the research project as 'practise runs', but these data were not included in the analysis. Nevertheless, some pupils may have talked differently because they knew they were being recorded and there may have been elements of 'playing to the gallery'. In addition there were some other unanticipated disadvantages of asking pupils to record their talk and these are considered in section 4.7 below.

4.5.4 Authored games

Twelve games were analysed to enable the researcher to identify the areas of learning evidenced in them. Using games as data gave the researcher information about what pupils achieved, as well as what they found difficult and where they had made errors. The analysis of the games involved scrutinising both the underlying static programming code and its output in dynamic game format (the games were run, played and evaluated). Yet the games do not represent a complete picture of what pupils learned, and may not accurately reflect pupils' understanding of the concepts involved, only what the pupils managed to do in the time available. Nevertheless, they provide information about what kinds of learning opportunities are afforded by programming a computer game (Denner et al., 2012: 242).

Using authored games as a source of data is increasingly common (see for example, Kafai, 1995; Robertson and Good, 2004; Pelletier et al., 2010; Baytak et al., 2011; Brennan and Resnick, 2012; Denner et al., 2012) and may be construed as an unobtrusive method of collecting data (Creswell, 2014) but has the following disadvantages:

- It is time-consuming, and relies on the researcher having in-depth knowledge of the software used to create the games.

- The games the pupils made will have been constrained by the affordances of the software and the scheme of work followed.

- The games may give some record of programming concepts used - but do not capture the thinking involved or convey the level of understanding reached.

- Content analysis is product-oriented, and reveals little about the process of developing projects, or the learning of something over time or the practices that might have been employed (Brennan and Resnick, 2012).

- As a relatively new data collection method the analysis of games authored by pupils is not widely covered in research methods literature.

4.5.5 Pupil documents

As sources of data, documents are relatively stable, less likely to be influenced by the researcher and more objective than other sources of data (Merriam, 2009: 155) and as outcomes of normal classroom activity the documents are less affected by the research process. They are, however, produced in a particular context (the classroom) and that will have some bearing on their content.

Pupil documents were of two types: planning documents created during class (initial game ideas, a storyboard, a design document, and outlines of the game interactions) and journal entries completed for homework (see Appendix 7). These documents represent pupils' responses to tasks, designed by the teacher, and these tasks will have both enabled and constrained the data that is collected in them.

Although pupils were asked to make journal entries on the same day of their ICT lesson, so that they would be able to remember their experiences more accurately, this was not possible to oversee and the reliability of these data may have been reduced by memory accuracy. Pupil documents were not sampled, yet they do not constitute a complete record, since some pupils did not complete all tasks, and there was also variability in the quality of the responses made. Nevertheless the documents yielded useful data about pupils' understandings of those elements of the game authoring process covered by them and their responses to the resources they used and the process they followed. The planning documents and authored games provide a concrete example of pupils' work which is supplemented by what they say about it in the interview and voice recording data.

4.5.6 Observation notes

Observation allowed the researcher to record behaviour as it happened (Merriam, 2009: 117) in the natural setting of the classroom. The researcher, as a participant observer (Merriam, 2009: 118) made ongoing, unstructured, descriptive field notes during the lessons, to record key lesson events, emerging problems (such as pupil absence), observations regarding pupils' responses to tasks (e.g. pupils' attitudes towards the planning stages) and comments and reflections regarding lesson activities and the salient features which arose (e.g. difficulties pupils experienced). In this respect it is a responsive method (Cohen et al., 2007). Observations were often derived from interactions with pupils in the context of normal teaching activities (e.g. explaining, demonstrating, answering questions, troubleshooting) and as such were non-intrusive. However, the reliability of these data is reduced due to its selectivity, which derives from the situation (Cohen et al., 2007: 398), as well as the fact that notes written up after the lesson were subject to memory accuracy. However, despite these limitations, the field notes further triangulate emerging findings.

4.6 Data analysis

The data analysis process in general was inductive and sought to establish categories and themes in the data (Saldana, 2011). These categories were then applied across the different data types. Interview and voice recording transcripts were analysed first, followed by pupil journals and planning documents; computer games were analysed last.

4.6.1 Use of NVivo 8 for data analysis

The data were transcribed and analysed using *NVivo 8*. For the purposes of this research, *NVivo* was selected because it can store and support the analysis of a range of multimedia data, such as that collected in this study (audio, text, graphic). It also enabled the collation of the data into a single file (pupil voice recordings and interviews (audio and transcripts), pupil documents, research notes, the coding system used and the analysis of the games). According to Yin, the production of such a 'database' strengthens the reliability of the data, since it can be used to replicate the study (Yin, 2009: 119). Using qualitative data analysis software can make qualitative analysis "more accurate, more reliable and more transparent" (Gibbs, 2002: 11) and can also contribute to a more rigorous analysis (Silverman, 2013). For example *NVivo's* queries function will find every coded instance of a concept, ensuring a more complete set of

data (Bazeley, 2007). The following sections illustrate how *NVivo* was useful for assisting the data analysis in this study.

Transcription

Audio files of the data collected in pupils' working conversations and pair and group interviews were imported into *NVivo*, and transcribed individually by the researcher. This increased the validity of the transcript since there was no remove between the researcher, the data and the transcription. It also enabled early knowledge building of the data (Bazeley, 2007: 44). As a transcription tool *NVivo* enabled greater accuracy because it was possible to change the speed of the playback to make unclear words easier to interpret. This assisted with the transcription of several pupil working conversations.

In total 8 hours of audio data were transcribed (61,093 words):

Group interviews - 1 hour 26 minutes (15,445 words)
Pair interviews - 2 hours 5 minutes (21,136 words)
Pair working conversations - 4 hours 28 minutes (24,512 words).

The transcripts were made immediately after the interview events, though there was a delay between some digital voice recordings and their complete transcription. The transcript was made with regard to ethical practice (Downs, 2010) and represents as near a verbatim record of what was said as was possible. The researcher included in the transcript pauses, mood indicators (such as laughter, emphasis, frustration) and interruptions. However, the chief concern was to capture the content of what pupils were saying.

Data coding

Using the research questions and the conceptual framework as a starting point, *a priori* codes were identified (see Appendix 5). Other 'open' and 'in vivo' codes were identified during analysis of the data to honour participants' voices (Saldana, 2011: 48).

Initial 'descriptive' codes were assigned to the transcripts of pupil voice recordings, interviews and pupil journal entries using *NVivo's* 'free nodes'. These codes were then thematically grouped into 'tree nodes' to capture a more fine-grained analysis. For example a free node of 'difficulties', was later reorganised to include the sub-categories 'design difficulties', 'narrative difficulties', 'programming difficulties'. Subsequently,

NVivo was used to generate a coding summary report used for the purposes of fine-coding specific difficulties within the main category 'programming difficulties'.

Each pupil was identified as a separate 'case', and gender and ability attributes were assigned. Ability was broadly defined in terms of pupils' Jesson band, based on the Data Enabler toolkit analysis (OFSTED, 2008) used by the research school. All transcribed data were coded to each case.

Following playtesting and analysis of the computer games, game analysis documents were coded for programming and design difficulties, using a further sub-set of codes (see Appendix 5).

Querying the data

NVivo's query tools supported the analysis of the data because they allowed the researcher to:

- Find all utterances made by an individual (case), for example to summarise what pupils had to say to inform the game analysis document created for each pair (coding query).
- Find all data coded at a particular node (coding query). Data coded at a particular node was collated to provide a framework for each of the findings chapters.
- Find all utterances which contain a particular word (text search query), for example, all utterances which contained the word or variants of 'enjoy', to inform the findings relating to affect.
- Find the number of times a particular word is used (word frequency query), for example the word 'fun'.
- Illustrate the results of a query in chart format (for example, see Figure 33).
- Run matrix queries, cross-tabulating gender and ability attributes with other code items. For example, ability was cross-tabulated with data that had been coded at the nodes for 'design difficulties', 'programming difficulties' and 'narrative difficulties'. The results of this matrix query, in Figure 10, show that pupils of high ability made more comments relating to these items than those of average or low ability.

	A : Ability = be...	B : Ability = ab...	C : Ability = av...	D : Ability = high
1 : Design difficulties	0	6	3	13
2 : Narrative	1	2	1	6
3 : Programming difficulties	2	6	3	15

Figure 10: Results of a matrix query

The same feature was used to explore the data for other queries, for example, the relationship between gender and references to difficulties (twice as many boys as girls reported programming difficulties). This was a useful feature of the data analysis software and was used to inform the findings reported in Chapter 8.

4.6.2 Analysis of authored games

The analysis of the authored games necessarily differed from the analysis of the interview transcript and other documents, in so far as coding is not as important for the initial analysis of visual materials as interpretation and analytic memo writing (Saldana, 2011: 82); the written record which grew out of the initial game analysis was subsequently coded.

A framework for the analysis of pupil authored games was constructed with reference to i) existing frameworks for the analysis of commercially produced computer games (Konzack, 2002; Aarseth, 2003; Juul, 2003; Consalvo and Dutton, 2006), ii) frameworks for analysing computer games authored by children (Harel and Papert, 1991b; Kafai, 1996; Kafai, 1998; Kafai, 2006b; Games, 2008b; Denner et al., 2012; Kane et al., 2012) and iii) documents defining generic computer programming concepts appropriate for Key Stage 3 and Key Stage 4 mainstream education contexts (e.g. OCR, 2011; Seehorn et al., 2011; CAS, 2012a; Edexcel, 2012b; NAACE, 2012; Saeli et al., 2012).

The analysis is in two parts. Part 1 (descriptive) deconstructs the authored games and identifies i) game design concepts evidenced in the games and ii) programming concepts used to construct the games. Part 2 (evaluative) applies a framework for a more holistic analysis of what pupils achieved in their authored games, and evaluates the learning they represent, as complete units of work. This framework was compiled with reference to relevant learning taxonomies (Biggs, 1979; Biggs and Collis, 1982; Hawkins and Hedberg, 1986; Fuller et al., 2007; Thomas and Martin, 2008; Thompson et al., 2008; Meerbaum-Salant et al., 2010).

Together these two analyses illustrate the analytic framework adopted and give a summative, holistic picture of what pupils achieved. The purpose here is not to assess learning outcomes in terms of normal student achievement measures, specifically, the National Curriculum levels of attainment (the attainment target for Key Stage 3 ICT was disapplied in 2012) - but to evaluate the overall *activity specific learning* evidenced in the authored games and in so doing, to address the research questions, 'How does computer game authoring using *Game Maker* support the learning of basic programming concepts?' and 'What difficulties do pupils have with game authoring (game design and game programming)?'

Part 1 i) Game design concepts evidenced in the authored games

In the first level of analysis games were thoroughly playtested and evaluated according to the criteria in Table 1 below, generated before the analysis.

Game design concepts	Definition
Usability	Game instructions, common control options (e.g. arrow keys), feedback, interface design, (design theme, animation), levels linked thematically/sequentially, more than one room/level
Functionality	Does the game work? Response to user input, interactions, gameplay
Graphics	Sprites, backgrounds, splash/title screen, animation, customisation
Scoring	Score, health, lives, high score table, rewards/penalties, win/lose states
Rules	Related to the program code, determines what the player can do
Narrative	Setting, story, character design, genre, non-player characters
Game design	The overall structural coherence of the game, object inventory, levels, obstacles, challenge
Cultural referents	The game's target group, its representations and cultural references
Sound	The use of background sound, sound effects or sound as feedback
Game goal	The purpose for playing the game (e.g. to free a captured character)

Table 1: Concepts used to frame the analysis of game design features

ii) Programming concepts evidenced in the authored games

The game analysis also measured programming concepts evidenced in the games, and in addition, the use of some mathematical concepts important for game programming (see Table 2 below). An initial analysis identified components of the game (sprites, objects, levels etc.) and counted and categorised actions and events used. The presence of the programming concepts below was then tabulated for each pair. The games were playtested again and a printout of the game information (program code) was annotated, to identify which elements of the code functioned as intended and what errors were made.

Programming concepts	Definition
Program interaction	Input/output, event driven. Are events used as input data?
Functions (actions)	Are actions used to create outputs in the game?
Sequence	Are events and actions sequenced in a sensible order?
Conditional statements	Are *test/check* actions used to test conditions?
Loops	Is the *step* or *alarm* event/*repeat* action used to create a loop?
Variables	Are variables (e.g. score, lives) used to store data in the game?
Logical operators	Are logical operators (AND, OR, NOT) used?
Boolean logic	Is Boolean logic (true, false) used?
Relational operators	(=, <, >) Are these operators used in expressions?
Mathematical operators	(+, /, *, -) Are these operators used in expressions?
Coordinates	Are coordinates used to specify screen position (x, y) of objects?
Angles	Are angles used to specify direction of movement of objects?
Negative number	Is negative number used (e.g. to define speed, position, score)?
Randomness	Is randomness used (e.g. to define position or number)?
Relative/absolute value	Is relative/absolute value applied to define score or position?

Table 2: Concepts used to frame the analysis of programming constructs

Part 2 Learning outcomes of authored games

After deconstructing the authored games to identify learning in terms of game design and programming concepts evidenced (a descriptive analysis), a more holistic, second level analysis was undertaken, to evaluate pupils' games as complete artefacts. This analysis looks at the overall structure and coherence of the game (i.e. the sum being greater than the parts) and is evaluative. It indicates the overall achievement evidenced by the game as a whole.

In order to systematically evaluate the qualitative achievement of pupils' authored games as part of the analysis, the two areas of learning highlighted in this study - game design and programming - were considered using the SOLO taxonomy (Structure of Learning Outcomes) (Biggs and Collis, 1982). The taxonomy describes 5 levels of response:

i) Pre-structural - the response has no logical relationship with the task, showing lack of understanding or inappropriate response.
ii) Uni-structural - the response demonstrates some limited understanding but may include minimal relevant responses or content.
iii) Multi-structural - responses are relevant but there may be no relationship between them or little internal coherence within the response.
iv) Relational - responses are related and appropriate and may contribute to a more coherent whole.
v) Extended abstract - the response is entirely appropriate and exceeds expectations.

This hierarchy of learning outcomes is applied to *particular* responses to a learning situation as "a means of classifying learning outcomes in terms of their [structural] complexity, enabling us to assess students' work in terms of its *quality* not of how many bits of this and of that they got right" (Biggs, 2003). Since this approach resonates with constructionist perspectives on assessment, which seek to evaluate learning outcomes qualitatively and holistically (for example, see Bruckman et al., 2000; Brennan and Resnick, 2012) and to "[measure] what kids can do with knowledge, not how many right answers they can give to questions" (Papert, 2001), it was adopted as an appropriate framework for evaluating pupils' games.

The SOLO taxonomy is also suitable for the current study because it is increasingly used to evaluate learning outcomes in computer science education (see Hawkins and

Hedberg, 1986; Lister et al., 2006; Fuller et al., 2007; Thompson, 2007; Sheard et al., 2008; Braband and Dahl, 2009; Meerbaum-Salant et al., 2010). Researchers suggest that a computer-science specific taxonomy is needed, because computing is a practical subject in which a key learning objective is the ability to develop computer programs, yet "the hierarchy of learning outcomes in computer science is not well captured by existing generic taxonomies" (Fuller et al., 2007: 153).

Accordingly, the researcher created an adaptation of the SOLO taxonomy (see Table 3), based on generic and programming-specific rubrics (Thompson, 2007; Whiteman, 2008), and oriented to evaluate the software-specific programming constructs, alongside the game design concepts, evidenced in the games pupils authored. This table was constructed after the final playtesting had been completed.

SOLO level	Score	Game design	Programming
Pre-structural no discernible functionality, no user interaction, graphics only	1/2	• no meaningful response - the game is not playable • no understanding shown • game assets (sprites, objects) may exist but are not organised or developed • irrelevant information • few, if any, interactions - the game is more a graphic or an animation • only one level exists • poorly executed graphics • no score mechanic	• no understanding of programming concepts • limited use of events • few actions implemented • events and actions are not combined effectively • no logical sequence • programming of simple instructions contains many errors • game functions minimally • little evidence that concepts learned in tutorials have been applied to the game
Uni-structural some functionality, some interaction, lacking development	3/4	• one aspect of the game is attempted e.g. the player character moves in one direction in response to user input • game assets (sprites, objects, sound) may exist but are not further developed • there are few game interactions • graphics may be poorly executed or inconsistent but are usable • no functioning score mechanic • progression through levels is not possible • a second level may exist but is incomplete	• limited understanding of programming concepts • events and actions are combined but these may contain errors • the game partially works with significant obvious problems • some of the concepts learned in the tutorials have been applied to the game • no use of conditionals • no use of variables • no use of loops

Multi-structural understanding evident, more functionality, a playable game, incomplete	5/6	• several aspects of the game are attempted e.g. player character and reward/penalty objects may exist but not all function correctly • game components are treated independently - connections are not made between them • some elements function correctly • a score mechanic is attempted but may not function correctly • some customisation of graphics - e.g. sprite change when player character changes direction	• the game includes several objects • the game includes more confident use of events and actions, some of which work • the game works with some problems • several concepts learned in the tutorials have been applied to the game • conditional statements are used, perhaps only partially correctly • loop constructs are used, perhaps only partially correctly • variables are used, perhaps only partially correctly
Relational all aspects cohere to form a playable game but some integration lacking	7/8	• an adequate response to the task • a playable game • several elements of the game are integrated into a coherent whole • the player can progress through levels • most elements of the game function correctly • graphics are more consistent and reasonably well executed • some customisation of graphics e.g. appearance change, animation	• a range of events and actions is used to control objects and operations in the game • the game works with no significant problems • programming concepts learned in tutorials are applied to the game • conditional statements are used correctly • variables are used correctly • loop constructs are used effectively
Extended abstract a fully operational game, coherent visually and functionally	9/10	• a complete, playable game with sufficient interactions to make it engaging • all elements function correctly (sound, score, objects) • the player can progress through levels and there is a clear win/lose state • graphics are consistent and fit for purpose • overall the game is a coherent whole	• experimentation with new ideas • use of programming concepts not explicitly covered in tutorials • use of execute code action

Table 3: SOLO taxonomy adapted to evaluate game design and programming concepts

The design and programming concepts in Tables 1 and 2 were reduced to 8 features (usability, functionality, scoring, game play, sound, game design, programming, graphics), and these aspects of the games were then evaluated using the SOLO levels

in Table 3. Each level of SOLO response was divided into 2 sub-levels, to give greater accuracy in evaluation (Chan et al., 2002). A qualitative 'score' corresponding to each SOLO level was given for each feature and an overall total was calculated for each game.

This evaluation was then represented as a matrix for each pair (see Figure 11 below), based on Thomas and Martin's model (Thomas and Martin, 2008) - an assessment matrix to evaluate hypermedia artefacts using SOLO levels on the vertical axis and key components being evaluated on the horizontal axis.

Extended	10								
	9								
Relational	8								
	7								
Multistructural	6								
	5								
Unistructural	4								
	3								
Pre-structural	2								
	1								
SOLO level		**Usability**	**Functionality**	**Scoring**	**Game play**	**Sound**	**Overall design**	**Programming**	**Graphics**

Figure 11: Matrix showing the SOLO levels applied to game features evaluated

These matrices show the achievement for each component of the game, the relative strengths of each game, and when viewed together, the qualitative variation between games (see Appendix 1).

Summary

This section has described the method used to analyse the authored games:

1. Games were playtested. An initial analysis was made, identifying components of the game and counting events and actions used.
2. Games were playtested a second time. Detailed notes were made about the functionality of the game. A written log of the programming code pupils used in their games was scrutinised. Programming constructs evidenced (see Table 2) were recorded in tabular format for each pair. The programming and design

difficulties they encountered/errors they made were coded in *NVivo* (see Appendix 5).

3. Games were playtested a third time to evaluate the game design elements evidenced - this analysis was recorded in tabular format using the concepts in Table 1.
4. The transcript of voice data and interview contributions for each pair was reviewed. Salient details were added to the analysis.
5. Design and planning documents were analysed for each pair. A separate analysis of these documents was completed (see section 4.6.3 below).
6. The games were evaluated holistically using a rubric based on the SOLO taxonomy (see Table 3). Levels of achievement for each component were illustrated using matrices.
7. A written description of each game was produced, taking into account all of the above (see Appendix 1).

This analysis forms the basis of the findings in Chapters 6, 7 and 8.

4.6.3 Analysis of pupil documentation

Eighty-five pupil documents were produced (plans, storyboards, journals). Journal entries were coded using the same codes as for the transcripts of interviews and digital voice recordings (see Appendix 5). Storyboards and work booklet data (initial ideas, design documents, game interactions plans) were analysed using a different method, since here data were presented in graphic or tabular format. A written commentary was made for these in which common themes were identified. These are reported in Chapter 6.

4.6.4 Analysis of observation notes

Observation notes for both pilot and main study were coded using the same codes used for the analysis of pupil voice recordings and interviews (see Appendix 5).

4.7 Validity and reliability of the data

4.7.1 Problems with data collection

During the data collection period there was a problem with attendance. Since pupils were working on their games in pairs, when a member of the pair was absent no voice recording could be collected. Absence also created gaps in the planning and journal

data. Absences were mainly due to illness, but there were also other unexpected absences due to uncalendared educational visits or events. Twenty-three individual absences were recorded during the thirteen sessions. No pair completed the project without absence.

In addition to absence, there were several issues which had an effect on the reliability of the data collected during lessons:

- The continuity of the project was interrupted by a half-term holiday which meant that there was a week-long interval between pupils completing the video tutorials and beginning their own game. This may have had an impact on some pupils' ability to remember the skills they had learned. Later on, a calendared 'Activities Week' also affected the continuity of the project, although by this stage pupils had gained greater familiarity with the software, and the concepts and processes of programming. Such interruptions (in addition to absence) may also have had an impact on some pupils' ability to work independently.
- Journals - some pupils did not complete all entries: 40/84 possible entries were made.
- Pupil planning documentation - some pupils did not complete all elements: 45/60 documents were completed.
- Digital voice recordings - the data did not capture a complete record of all pupils' working conversations; the data recorded by pairs varied in amount considerably - range 0.5 minutes to 58 minutes, as shown in Table 4 below. The two pupils who worked as individuals did not produce recordings.
- Saving voice recordings correctly to the network - some pupils saved more sound files than others.

No. of mins	<10	11-20	21-30	31-40	41-50	51-60
No. of pairs	3	1	2	1	1	2

Table 4: Digital voice recordings of pupil working conversations

Although there was an expectation that pupils would record their working talk, some pupils were less concerned to do this and more interested in making their games. There was a feeling that 'we've recorded loads already so we don't need to record any more'. Additionally, an unforeseen outcome of this method of collecting data was that

pupils ‘sanitised’ or ‘prepared’ their talk for recording - and were disinclined to record the normal ‘messy’ speech of their working conversations.

Despite these gaps in the audio and written data and in the levels of completeness of the authored games, 17/22 pupils were represented in either the pair or group interviews; 12 games (10 pairs and two individual) represented a complete record of all the games made; 85 pupil documents were also analysed.

The validity of the interview data may also have been compromised because the researcher was not able to return the transcript to the pupils to verify if it was an accurate record of what they said, since the interviews had been transcribed during the summer holiday after the data was collected and the researcher no longer taught the group. Logistically it would have been impractical to invite pupils to review the transcript given the interval between the fieldwork and the new school term.

4.8 Ethics

The ethics protocol for the field study period was approved by the supervising university’s Research Ethics Committee. The teacher/researcher addressed the ethical issues arising from the power differential between herself and the pupils as described in section 4.5.2 above. A risk assessment considered the social welfare and the health and safety of participants. The research interest was declared to the Headteacher of the research school and permission was granted for the field work to be conducted with pupils and for the collection and storage of data on school computing equipment and the network.

The research interest was declared to pupils orally and in writing at the start and orally throughout the data collection period. Pupils were asked to give written consent for their work to be included in the research study. An information booklet was prepared to make pupils aware of the research intentions and purposes, and pupils were asked to sign to confirm that they agreed with a set of statements relating to the collection and uses of the data gathered. A letter for parents conveyed similar information, and asked parents to give their written consent to their child’s work being used as part of the research study and to confirm their agreement with the same statements (see Appendix 2).

Pupils and their parents were assured of the confidentiality and security of the data

gathered and informed about the uses to which the data would be put. Pupils were made aware that they could withdraw from the research activity at any stage, in which case their data would not be used, although there was an expectation that they would complete the project tasks, since this was part of 'normal' classroom activity.

Data protection

Pupil data were stored securely on the research school network and analysed on the researcher's personal computer. No third party had access to the data. Access to the data on the school network and the researcher's personal computer was password protected. Data were deleted from the school network after they had been collected. Pupils' identities and the identity of the research school were anonymised in the thesis.

4.9 Summary

This chapter has described and evaluated the research methods selected (case study) and the instruments used to collect the data (interviews, voice recordings, documents, authored games). A description of the data analysis methods used is given and the ethical issues surrounding the research have also been considered.

Chapters 5-9 following present the findings and analysis of the study. Chapter 5 describes and analyses pupils' perceptions of the process they followed in constructing their games. Chapter 6 investigates the areas of learning and the difficulties they encountered in terms of game design. Chapters 7 and 8 examine the programming concepts pupils used and consider the difficulties they experienced with these. Chapter 9 records particular values of the game authoring activity, beyond the learning of curriculum content, identified in the interview transcripts and observation notes. A discussion of the key findings and implications of the study is presented in Chapter 10.

Chapter 5 Making games – the process

5.1 Introduction

This chapter documents pupils' perceptions of the process they followed as they made a computer game. The activity had been organised with regard to the principles of constructionist learning theory, and pupils' responses to this approach shed light on the extent to which constructionist practices are suitable for the different activities involved in game authoring. The findings reported here refer to the 'eight big ideas of constructionism' outlined in Chapter 3.

Data were gathered from digital recordings of pupils' working conversations, pair and group interviews and journal entries and coded for references relating to the working processes and practices followed.

5.1.1 A constructionist designed activity

Pupils worked with a partner to design and make a computer game. In the early stages of the unit of work, pupils researched and planned their ideas, and followed structured video tutorials to learn how to use the software. Once underway with making their own games they entered an exploratory learning phase, where they gained greater familiarity with the software and the process of making games through learning by doing and experimentation, key components of constructionist learning theory.

5.2 Resources

In a constructionist learning environment design activities play a central role (Kafai, 2006a) and different media and activities are combined (Kafai, 1995). In this research, multimodal materials were selected to appeal to a range of learning preferences:

- The software made available to create the game enables (inter)active learning and functions as an 'object to think with' (Papert, 1980b: 11).
- Resources to support the activity included a pupil work booklet, which provided materials to support lesson activities, rubrics for planning the storyline for and the rules of the game, a test plan, and a glossary of events, actions and useful software commands.

- Sample games illustrated the possibilities of the software and gave pupils access to the code used.
- Text based tutorials exemplified how to create different genres of games.
- Online video tutorials introduced pupils to the basics of the software, and of the game-making process as they were shown, step by step, how to create three, increasingly complex games.

While most pupils found elements of the resources useful, there was a feeling that they were either limited or constricting in some way, and this is significant - the provision of suitable support resources is of key concern when pupils are working independently on individual extended projects, since it emerges from the literature that game authoring is not well suited to linear delivery (see Willett, 2005; Robertson and Howells, 2008).

5.2.1 'Just in time' learning

The notion of 'just in time learning' (Riel, 1998) is important here. Observation notes record that pupils wanted sources of support which would provide solutions to their individual problems at the point of need and guidance for how to implement common game mechanics, rather than the linear game tutorials made available:

> AE: [It would be useful] if there was, like, a sheet which had, like, all the [events and actions] and each thing told you what it did and an example of what you could use it for.

'Just in time' support was also needed because it was difficult to maintain continuity during the game making activity, which was delivered over an 8 week period, due to timetabling patterns, pupil absence and other school imperatives, such as half-term holidays and an activities week. Some pupils found it difficult to remember the content of the tutorials they had followed, since there was an interval between the tutorials and the start of making their own games. Others found it difficult to transfer what they had learned in the video tutorials to their own game or did not transfer their prior learning in graphics to the game context.

Efforts to access 'just in time' support via the *Game Maker* website or online tutorials and manuals were thwarted by the research school's internet filter:

> JC: We really needed the … Yoyo Games website. Because you [had] to, like, create the sprites yourself. It's quite hard to find all the sprites you need for the game without it.

Other websites provided definitions or specific tutorials, but did not provide the sort of particular help that pupils needed for their individual projects and many were not written in a pupil-friendly style.

Having access to 'just in time' resources was also important because some pupils resisted following the semi-structured scheme of work offered. Pupils wanted to 'learn by doing' and make their games. Once begun, the process of creating their games became 'immersive', to the extent that some pupils viewed any whole class instruction as an unwelcome interruption. The scheme of work sought to guide pupils through the systems development design process, and introduced focal activities for each lesson - but pupils were reluctant to interrupt the flow of their computer-based, practical work to attend to these. These factors in combination make access to 'just in time' learning resources important.

5.2.2 Learner control

The pupils in this study expressed a preference for materials which gave them freedom and control over their learning, even when these were not highly visual or interactive. For example, MD preferred to use textual resources rather than the video tutorials, because he felt he had more control over text:

> MD: [The video tutorial] goes at a set pace, whereas text you can just stop … I liked the text based [tutorials] because they had pictures to show you … what the game should have been like. They had a lot of text and it went into a lot of detail and it told you exactly what you needed to do, but it also left you to be a bit more creative with what you wanted to put in.

Textual tutorials enabled this pupil to be more creative, because they allowed him to apply the learning to his own particular situation, whereas the video tutorials only showed the techniques needed to make the particular game featured in the tutorial, which was perceived to be constricting:

> MD: It told you only how to make *that* game, whereas in the text based ones you could sort of use what's in the tutorials to help you make your own one more easily.

However, others did not use the text-based materials:

> AC: We had the booklets here and they were just a bit of a distraction I think.

> JB: I looked in the book, but from the book to the [video] tutorials, I prefer the computer ones 'cos … I [learn] by trial and error and listening and watching, and I don't tend to take in much when I'm reading as well as I do when I'm listening.

> SA: I remember having the books, but me and R. just used the [video] tutorials.

Others criticised the video tutorials because they did not support pupils in their individual endeavours:

> AC: We only learn the basics of all the controls, we didn't really learn how to do more advanced things.

> GW: It helps you with the first few bits then afterwards you get really stuck.

> KW: They didn't teach you how to get onto the next level so I didn't know how to do that.

> MD: I didn't really like the tutorials ... They were a bit patronising I think and they were very slow to get to the point … I think most people did find the video ones a bit boring because every time you finished a tutorial it then spent about 5 minutes recapping on the previous one, which was a bit tedious.

Whilst there were several complaints about the audio which accompanied the video tutorials, others liked this mode:

> MH: I like the idea of listening to that lady talk about ... that was easier 'cos you could go through the steps while she was talking.

> LW: You could pause the tutorial as well, when you needed to, so that you could complete yours step by step at the same time … you could go at your own pace, like you could rewind and whatever.
>
> JB: [The video tutorials] make learning more fun and easier to do. The online lessons are better than books.

Some pupils perceived the video tutorials to be useful because they could *see* the processes they needed to follow:

> KW: It's just a different way of … teaching you, 'cos … if you're, like, visual then it's more helpful for you 'cos you can see the different things in front of you, also you're more independent.
>
> SA: If there was a choice between textbook and watching a video tutorial I would pick the video because they're easier to, I mean when you visualise it, it's easier to put it together yourself.
>
> MD: You can just … change between windows, so you can see the tutorial while you are making the game.
>
> JBr: [The video tutorials were useful] if there was something more advanced that … you actually needed to *see*, rather than just like read.

Pupils' comments about the affordances and constraints (Kennewell et al., 2007) of the resources available to them underline the need to provide a range of multimodal resources to appeal to different learner preferences. Such resources should enable pupils to find solutions to common problems, give examples of when particular events and actions can be used, and model the code for common game functionalities; however, none of the available resources provided this sort of targeted, modular support.

But their responses also highlight the need for some pupils to develop a different approach to learning. Whilst they valued the freedom of working in a constructionist learning environment (learning by doing/making, working in pairs, choosing their own tasks and deciding which resources to use) that way of working brings its own demands and challenges pupils to be more proactive in their approach than some are

accustomed to. This finding leads in to the next section, which considers the extent to which the game authoring activity provided an opportunity for pupils to 'learn to learn'.

5.3 Learning to learn

One of the tenets of constructionism is that working on projects over an extended period of time enables pupils to 'learn to learn' because they come to view learning as a process, where different styles of working or approaches to tasks are equally valid, where errors are inevitable, and important sources for learning, and where the real work consists of 'tinkering', making refinements and frequent 'debugging' (Papert, 1999a). The findings described in this section show how the game authoring activity gave pupils the opportunity to learn to learn - from each other, by doing, by trial and error, by interacting with their games in progress, and for some, by using the software itself.

5.3.1 Objects to think with

Game Maker supports self-directed learning in terms of its accessible drag and drop interface, graphical icons, help menu, visual feedback and the availability of its inbuilt sample games, resources and online community. But pupils were not accustomed to making independent use of this 'web' of supporting resources. Using the software and surrounding resources as a source of support was a new way of working and they needed guidance and prompting to adopt this approach. Whilst they needed 'just in time' resources so that they could access "knowledge in use" (Papert, 1993: 63) they also needed the wherewithal to make good use of them.

Visual programming environments such as *Game Maker* are referred to as 'objects to think with' (Papert, 1980b: 11) or 'mindtools' (Jonassen, 1996; Meijers, 2012), and, it is argued, using such software engages pupils in higher order thinking and encourages them to reflect on their learning because it enables pupils to externalise their ideas and thus makes their thinking more visible to them (Papert, 1980b: 11). However, in this study, the extent to which pupils used the software in this way was variable. While the software was a 'learning space' (Zorn, 2008), and an 'object to think with' for some pupils (AEMD, ACJC), for others it did not have the same appeal.

Indeed, most pupils did not make use of the sources of support within the software, such as the *Help* menu, and did not explore the other menu items or features available. Whilst some pupils learned how to do new things by experimenting with interface

options (AEMD used the debug mode to help them isolate errors, for example), others did not. There was a reluctance to read textual information provided in the software and pupils did not read or respond to error messages or textual hints (KW, CB). This suggests that the pupils in this study were not used to working with software in this way and had not yet developed skills in seeking out solutions to problems independently within the software, or lacked the confidence to do so because they were using new software.

Although most pupils did not actively explore the software, part of the process of making their games was led by a 'create by reacting' approach (Victor, 2012), where pupils reacted to the functions and components available in the software. For example, one pair (AEMD) explored the 'global game settings' menu item and by doing so learned how to customise the loading graphic, game icon and screen size of their game. This way of working typifies constructionist approaches and for some became the dominant way of working as they interacted with their games under construction, trying things out or solving problems.

Game Maker provides some support for this type of approach. Graphical icons give visual clues to the behaviour of an action and textual hints which appear on mouse rollover give further support (see Figure 12).

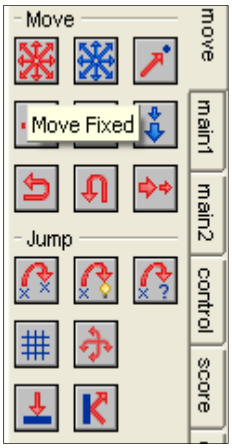


Figure 12: Graphical icons showing textual hint

However, not all pupils found the iconic representation of actions intuitive:

> CB: I didn't know what any of them did or how to use them, so I didn't know how they could help me improve our game … the symbols in each box aren't very clear so it's quite hard to find each different action needed.

Textual hints appear if an icon is hovered over, but these were not always perceived as useful:

> CB: I didn't know what it meant.

For some pupils, this 'not knowing' did not lead to them trying to find a solution independently. Their readiness to 'learn to learn' was not always evident. Perhaps for them the abstract programming environment was difficult to explore because it offered too many options.

5.3.2 Learner-directed activity

The pupils in this study relished the relative freedom they were given to direct their own activity and were, to varying degrees, successful in managing their learning for the duration of the project. Their enthusiasm for learning by doing meant that once underway with their projects, some pupils were resistant to teacher intervention and although they continued to use the teacher as a troubleshooter and resource, they preferred working practices which did not rely on teacher input:

> GW: I liked it that you … could come in and start getting on straight away and you taught yourself rather than watching.

> JG: We learned for ourselves, although we had to go onto that web site, the tutorials, and learn by ourselves, but we normally do it as a big class.

> MH: We had more freedom learning from a video tutorial … I think it would be good to use these again because they make us more independent so we don't always need a teacher.

> GW: It's better not learning off the whiteboard, 'cos I have a really low concentration and I just sort of switch off when it's being explained on the whiteboard.

> JC: With our normal things we do in ICT it's just 'Do this, do that', and yet in this one you actually get a chance to sort of find out for yourself and do it yourself.

> OW: The tutorials are good because when your teacher explains the lesson, afterwards if you are stuck or unsure what to do you have to wait until they

> come over and explain it to you. With the tutorials when you are stuck you can just flick back in the video and see where you went wrong without disturbing your teacher. This also helps us to be independent while working on computers.

This reference to independence is significant and seems to be related to an increased sense of self-esteem, confidence and self-efficacy, which arise out of self-directed learning, attributes which have been widely reported in the literature in constructionist practice generally and in authoring computer games in particular (e.g. Harel, 1991; Kafai, 1995; Sanford and Madill, 2007a; DEECD, 2010; Li, 2010).

Learning to learn involved not only selecting resources but also organising and sequencing their work. For most pupils this meant that what they did in each lesson was determined by what they had done in the previous lesson and the issues which arose as they developed their games:

> JC: Well you don't really tend to look ahead to it. You just sort of, when you get there you look at what you still want to do and you keep clicking on the [play] button to view your game and see how it's developing.
>
> KW: Well I just looked at what I had and then I realised what I needed to do and I just did it … When you started it was a bit like 'Oh what am I doing now,' but then once you got used to working out what you had to do it was fine.
>
> AE: Right at the beginning of the lesson we just, like, allocated tasks for each other and then just got on with them.
> MD: Well, we just sort of went through in a logical order.
> AE: We just sort of thought of a plan at the beginning. I will make level 1 and M. will do level 2 and then…
> MD: Yeah ... we'll see who gets finished first to do level 3.

The software itself supported some pupils in managing their work. For them, the resource explorer (see Figure 13) functioned as a visual reminder of the components needed in a game and the order in which tasks should be completed:

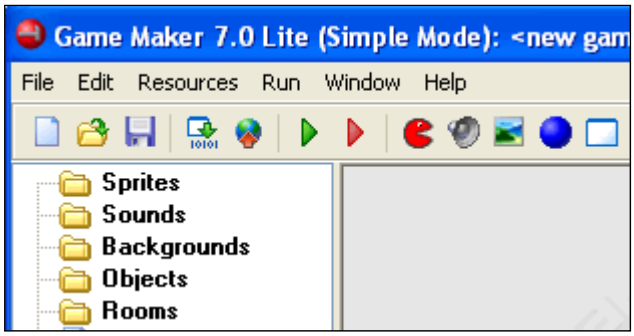


Figure 13: Game Maker's resource explorer

JG: We just looked as we were going, step by step. We would obviously get our sprites and objects and then we made our backgrounds. We went step by step for what you actually have to do.
JB: 'Cos the files ... we went down the files that are there and then the room and we started ...

Others were led by the narrative of their game:

JD: Well it's pretty obvious at the beginning, 'cos you need to make a background don't you, to have a game, so we started off with that, so we just started off just making all the sprites work, put them into objects and making the main character, he has to work first and what the main character does. We did that, and then just like the obvious ones and then the little things come next.

The data here indicates that in organising their own work some pupils preferred to be "guided by the work as it proceeds" (Papert, 1991b: 6), a way of working which according to constructionist theory is as valid as more formal structured approaches, even though it may not have been efficient:

MD: I suppose we probably could have made a bit more effective use of our time. We did sort of spend a bit too long on the fun things …

AE: We spent a bit too long making the sprites and making them look good and animated.

But while some pupils enjoyed the fact that the activity was not teacher-led, others were not so enamoured of having to direct their own learning:

> CB: I didn't know what to do lesson to lesson. I didn't know how I could take my game to the next level, like keep on improving it.

> MH: A. said that it's always fun and challenging but I dunno. It was fun when we knew what we were doing, but I got really annoyed when we were just sat around thinking 'Oh what do we do now?' and it wasn't as fun.

5.3.3 Styles of learning

Papert refers to two styles of learning: 'top down' (planners) and 'bottom up' (bricoleurs) (Turkle and Papert, 1990: 136). Pupils in this study used both. Some pupils favoured a 'bricolage' approach (Papert, 1980b: 173) and resisted attempts to direct their learning. Others were more methodical, and tended to have more success both with the process and outcomes of their learning:

> MD: I think we had quite a clear view about what we were going to do each time, 'cos we knew exactly how we wanted our game from the beginning, so if you know that … you know what you want to do every lesson.

This contrasted with others' experience:

> CB: I think if you don't know what you want then it's much harder to get it. 'Cos we weren't very specific on exactly how we wanted it to be and you said that you knew how you wanted it to work and what you wanted it to look like and stuff. That was like the difference between us. We changed our minds as we went along. Yeah we weren't specific enough.

> SA: Yeah, that would have probably been our problem, 'cos me and R. weren't very clear from the beginning what we wanted our game to be like ... we just sort of put it together as we went along.

Here pupils make specific mention of how their lack of clarity at the design stage, in terms of how they wanted the game to work and what they wanted it to look like, caused them problems with the implementation of their games. For these pupils, being guided by the work as it proceeds was not so successful. Others pondered whether more project milestones would have been useful:

> MH: I think it would have been good if ... at the beginning you know where … we did that [Gantt chart] for how many lessons it would take to plan and then design. I think we should have done that … and then I think that would be good to have like a guide, but then on the other hand I think it was good that we could work at our own pace.

For the pupils in this study, taking control of their own intellectual activity in an extended project was a new experience and whilst they were able to direct their own learning in some aspects of the game authoring activity, in others a more structured approach may have been more supportive. These findings suggest that bricolage is not an effective way of working for some pupils and some tasks. This gives support to other studies which have found similar difficulties with self-led learning (e.g. Bruckman et al., 2000; Reynolds et al., 2010).

5.3.4 Pupil journals

In order to encourage reflection on their experience of making a computer game, pupils were asked to keep a written journal as a weekly homework task (see Appendix 7). The data collected here suggests that pupils found it difficult to reflect on their learning in this way, and in keeping with their preferences for learning by doing, some thought that the time would have been better spent if practical tasks to do with game creation had been set:

> MD: We could have done homeworks which actually included working on the game.

> JB: It was like ... like with the question ones, it was … 'What did you find difficult about it?' And I was like, 'I found this difficult' and then 'What problems did you have?' 'Oh I had problems moving the horse and I didn't sort it out in the end' and then it was like 'Oh so how do you sort it out?' And it was like, 'Um, we haven't managed to yet'.

This disinclination to write about the game-making process is echoed in other studies (e.g. Kafai, 1995; Sanford and Madill, 2007a). In practice the journals were ignored by some pupils; they wanted to make their games, not write about them. Entries were descriptive rather than reflective, and this suggests that more structure needed to be given for this task. A greater focus on program annotation may have been a more suitable site for reflection.

Nevertheless, journal entries made demonstrate that some pupils were 'learning to learn' in so far as they write about adopting a range of practices in their work, some of their own devising:

> LW: To decide the rules, events and how you get points we made a table with two columns. We then listed the ways that the [characters] could lose lives and gain lives so that we could do a near enough balance, making the game successful and not easy but not impossible, giving both objects a chance.
>
> MH: C. and I took great time deciding on our game ideas. We planned and thought it through, and even when we thought we had a good story line, we decided to re-think and make it more suitable for year six pupils. We drew out a thought bubble, whilst speaking into the voice recorder and suggesting our ideas. We then chose the best ideas for each 'element' of the game ... To decide on the rules of the game we looked at and played on some examples on the internet to see how they worked and the sort of rules that applied to them.
>
> LW: G. and I came up with our game, Squeek, by researching other games that are highly rated. This gave us an idea on what kind of games people liked playing. We found out that animal games were firm favourites and mazes. To help us with this we thought about the kind of games we liked when we were of that age and asked others to share with us data on what they enjoyed playing to help us with our game aim and story line.
>
> GS: I decided on the rules of my game by playing other games … I have looked for ideas on the sample games, such as the Pac-Man game and the car game. These helped me out a lot and gave me ideas on how to do improve my game.

These examples show that, given the freedom to direct their own learning, some pupils made use of strategies to help them organise their work (tabulating ideas, using thought bubbles, playing other games, consulting others). In terms of learning to learn, the extended nature of the project gave these pupils an opportunity to explore their own ideas for learning and to develop individual strategies over time.

5.3.5 Learning by doing

A key principle of constructionism is that people learn best by doing (Papert, 1991b) and the findings reported here show that most pupils in this study preferred this way of working. Some pupils expressed frustration when they were asked to stop practical activity to plan or to attend to a teacher demonstration:

> AC: You can just do all this in *Game Maker*!

> JC: [We should] focus more on making the game, rather than other things like the booklet.

> OW: I like doing the practical side of things and I prefer it to listening to our teacher instructing us, because that restricts the time on computer and if we are on the computer right from the start then it helps us improve our practical skills on the computer even more.

Learning by doing also seems to have a positive effect on engagement for some pupils:

> JB: It's just another way of learning to make it a bit more fun, rather than normal reading out of a book, so it gives us a chance to try and do it.

> KW: It gives you more independence and you get to choose what you're going to do and be more creative.

> JBr: I liked the game making 'cos I don't tend to like reading the instructions, I like to experiment with it and with the tutorials I sort of listened a bit and then when I went on [*Game Maker*] I was just exploring with it so I found a lot more things without like needing the tutorials, so ...

> JC: You get to use your memory more doing it yourself as well, rather than just copying something.

> JG: It makes you think more than the plain work that everybody does, like, simple Microsoft work. This one makes you think really hard about what you're doing.

This preference for computer-based practical learning accords with constructionist learning theory and gives support to other studies using different software - where students preferred to have as much hands-on computer time as possible when creating games (see Sanford and Madill, 2007a; Werner et al., 2009; Smith and Sullivan, 2012). However, those studies found it necessary to provide guided challenges to frame practical activity to focus pupils' learning on the skills required to build their games. The key finding here is that while most pupils in this study preferred to learn by doing, and were less inclined to listen to teacher instruction or demonstrations, guidance needs to be embedded in the practical work they do if important concepts are not to be missed. This highlights a need to provide more contextual support in open-ended projects so that a balance is achieved between enabling pupils to learn by doing and ensuring that they follow a path which covers the learning objectives of the unit of work. This finding is supported by several research projects which aim to address this need either within the software, or by strengthening collaboration within and beyond the classroom (e.g. Good et al., 2010; Frydenberg, 2013; Ahmadi and Jazayeri, 2014).

5.3.6 Freedom to get things wrong

Another of the key ideas of constructionism is that, as part of learning to learn, pupils need to be given the freedom to get things wrong and learn to view errors as a source of information (Papert, 1999a). The pupils in this research encountered many problems (see Chapters 6 and 8) but as in other studies of making games (e.g. Li, 2010), the problems they encountered did not seem to deter them. The most common approach for solving problems was through trial and error:

> SW: OK, at the moment we're trying to do it so on our start screen, [when] we press the start button, it goes straight through to the game. So we don't actually know how to do that right now so we are going to do it with trial and error.

> AW: So T.'s just editing ... the crate because it had a green background instead of transparent and we're not sure how to fix it, so T.'s just trying anyway he can.

Papert does not advocate 'trial and error' learning - and refers to it as 'slow and primitive' (Papert, 1980b: 113). He suggests that more can be learned if pupils begin to analyse their thinking, and develop strategies for 'debugging' program errors. The data shows evidence that some pupils did begin to think analytically, or adopted strategies for solving errors, built on their current understanding:

AE: The debug mode was useful because then you could see ... it came up with an error and put it in an error log.

KW: Well I just used common sense, so if [the problem] was something to do with the score, I would just go onto the [actions] and look around for something that might help and I would test it.

JB: Right, okay. We've just managed to make the screen scroll. Hang on, how're we going to get it to go the other way? Oh I think we need to do minus ... minus 6, let's try that.

TB: Well there was another game that was an example which we could look at and that gave us an idea of what it was, but then we just sort of tried different combinations and tested them to see if they worked or not.

AE: We've made an animation for the intro but when we insert it into *Game Maker* it changes the frame rates, so we are trying to get around this problem by changing the animation length … I set the frames to 140. I wonder if that will work … Well it sort of works, it's better than before isn't it?
MD: Yes but I think we can still change it a bit more to improve it.
AE: Maybe I should try a different number of frames.
MD: No I don't think that would work, 'cos we need the difference between the flashing of the logo ... to be different.
AE: I know. If we just insert lots of extra frames that are the same, it will look as if one of the frames is longer.

Whilst a trial and error approach was sometimes successful in solving problems observation notes suggest that it was not efficient and sometimes led to frustration. The data here support Papert's assertion that pupils need to be taught to view errors as sources of information and to develop a more strategic approach to solving errors. For most of the pupils in this study that would involve learning how to respond to error messages and making greater use of the sources of support provided in the software (e.g. using the *Help* menu), as well as checking and testing the programs they created more systematically.

Whilst 'out-of-school' approaches to learning through trial and error and repetition are reported to be successful with children's use of computer technologies in the home

(Downes, 1999) data from the current study suggest that when making games in a school setting, such approaches are successful only to a limited extent. Research initiatives such as the Flip project (Good, 2011), where software has been developed to provide contextual support as needed, offers a promising solution to this problem. Using collaborative learning tools, blogs and social media and encouraging peer learning are also useful strategies (see Frydenberg, 2013; Ahmadi and Jazayeri, 2014) to support problem-solving in open-ended projects such as these.

5.3.7 Working in pairs

Collaboration is an important feature of constructionist learning environments (see Kafai and Harel, 1991; Harel and Papert, 1991b). In this study, the process of game making was shaped by the collaboration between pairs. Although pupils will have worked with others previously, it was not common in the research school for them to co-create interactive digital artefacts. Pupils generally enjoyed working in pairs and found it enabling:

> JB: I liked the way we could work together … because most people find it easier to work with a partner.

> CB: 'Cos you get to put together your knowledge … because if you don't know how to do something … they show you how to do it. And also you have more ideas because you're not the only one thinking of ideas and their ideas help.

> SA: I think it's much better than working on your own because if you feel stuck you can just ask your partner and they might know or ... and also it's easier.

> GW: If you're working in, like, a pair, you don't have to go and ask the teacher's help; if you're stuck it's ok.

Pupils here seem to value working with others because of the 'unobtrusive' support it offers (Harel and Papert, 1991b: 42). Working in pairs was also perceived to be useful because partners provided useful critical feedback:

> AE: It was good because we had more ideas and we could, like, combine them both to get a better ... project.

AC: When you are creating your game and thinking up ideas for it, if you think up an idea and you think it's quite good, your other partner will say this is what's bad about it and this is what you need to change and this is what's good about it. Which is a bit different. You know, if you're on your own you'd probably create a worser game.

MD: Yeah, we could see what wasn't going to work and what was going to work and if you've got a second opinion about something then it helps a lot.
AE: Yeah 'cos you can have constructive criticism.
MD: Yeah and generally it's good to have a second opinion about something that you're making. Someone else can see the faults in what you're doing or think up new things.

Pupils also valued the flexibility of being able to work with others and to pursue their own ideas and tasks within the shared enterprise. So while it was important to work with others in terms of planning and problem solving, pupils also wanted to be able to complete tasks individually according to their interests. This was particularly the case with creating graphics, or when pairs decided to work on separate levels in order to progress the game. The 'optional' collaboration seen here is an important feature of constructionist learning cultures (see Kafai, 1995: 294).

Although pupils liked working in pairs, it also created some problems:

JD: Yeah, if you have an idea and you think it's really good and then your partner has another idea and you can't decide what idea to ... Say you're thinking of a story line and your partner doesn't like your idea and you want your idea, then you'll probably have an argument.

JC: We did have a few arguments.

JG: As I am making my game with J. there has been lots of argument, disagreements and problems. Some of the problems me and J. have had are not agreeing on the game name, agreeing on the characters. To solve all of those problems we had to keep thinking of different ideas until we both agreed on it.

> TB: We had to come up with a sort of realistic storyline which would keep the audience interested and it was hard because we both had different opinions on what that was so ...

> MH: We have encountered a few problems along the way, such as deciding what our final, approved story-line was going to be. After many discussions and negotiations, we have finally decided on a suitable story-line.

Learning to manage these differences of opinion is, in constructionist contexts, part of learning to learn with others. The above comments illustrate that pupils had to learn to negotiate and reach agreement, particularly with the early project tasks of planning and initial ideas development.

5.3.8 Learning from others

Whilst most pupils learned with and from their partners, some also learned from others in a wider sense when they reused code from sample games. Several pairs (AEMD, AWTB, JBLA) made use of elements of code in sample game files or tutorials to help them understand and apply new programming constructs in their own games:

> TB: Now we've got an idea on how to do it, we'll copy the 'up' button [code from a sample platform game] and see if that works for our character and if it does that'll be a brilliant breakthrough because then we'll know how to do a lot of the rest of it.

One pair (JBLA) used a script they located on the *Game Maker Community* forum. Pupils also used peers as a resource. For example, AEMD correctly implemented the code for making an object reappear once it had travelled off the screen and this was reused by other pairs (OWSW, JBJG). As in related studies (Good and Robertson, 2006a; Gross et al., 2010; Smith and Sullivan, 2012), there was a 'brushfire' effect, where pupils reused code created by others and in so doing taught each other new programming constructs. In this way pupils learned from 'more knowledgeable others' (Vygotsky, 1978) as they began to develop their own community of practice (Wenger, 1999).

> JB: On our game, 'cos our background was grass and sky, we had loads of trouble trying to get [the horse] to stop at the grass and stop at the end instead of going off screen, so we, like, we ended up getting some help from A. and M.

> TB: Yeah because when you click ... on our instruction button it should come up with text but we don't know how to tell it to do that so we're just trying to work out how. Or I'll ask M.
>
> CB: [I would] just look at what other people had done so far and find out, like, what was missing from ours which we could have added to.

This 'collaboration through the air' (Kafai and Harel, 1991) is a positive feature of constructionist learning environments, where pupils are immersed in a shared learning activity and have more freedom to interact with others and others' ideas. Because by their very nature, games exist to be played, the authored games were shared with others, in a way that perhaps outcomes of conventional ICT projects (for example, a database) are not. Observing and providing feedback on each other's games gave rise to new understanding and ideas and encouraged pupils to add similar features to their own games. This observing others, imitation, peer teaching and knowledge exchange illustrate how collaboration in the classroom was heightened by the game authoring activity and the constructionist approach.

5.3.9 Taking time

One of the central tenets of constructionism is that learning by making takes time (Papert, 1991b) and that giving pupils time to complete projects is necessary if they are to become personally involved in their learning. Pupils in this study were given 16 hours to make their games, although none of the pairs completed their game in that period. Nevertheless, working on an extended project had a positive impact on the way some pupils in this study approached their learning:

> AC: It makes you concentrate more 'cos you're always doing something, and in other IT work we just come in and sit down and we listen to the teacher and they explain stuff on the whiteboard and then we go off and do something on the computers. But when we are creating a game we come in and sit down and we get on with our game straight away.

In fact, pupils seemed to value being able to work on an extended project and some would have liked more time:

MH: I just think we should have had longer to do it as well … and then we could have spent the last few lessons looking at people's finished games and done more evaluating as well. I think that would have been good.

KW: Yeah I think we need longer to finish off our games but I think we should have spent less time planning, because we should have spent two lessons planning and then got on with it.

CB: I think we didn't have enough time to do the, like, section of game making because, well yeah, we haven't finished our games. And it would have been better to learn more about it and how to do it before we actually started making them.

MH: I think we should have made … you know that piggy one we did? I think we should have done something like that in an earlier year, like, before, so that when it came to this year we could get on with more advanced stuff, because a lot of us had no idea at all how to do it, so we spent quite a lot of time learning the basics.

There was a feeling that lack of time was a limiting factor:

MH: I think we had our expectations too high, then we realised that we didn't have that long to do it so we had to make it simpler.

MD: I don't think anybody finished as to what they really wanted their games to be like. 'Cos we got ours working sort of, but there are still quite a few bugs in it.

SA: Yeah I think if we'd had longer we could have made our game better.

AC: It's just that you've got so many ideas and you've got to incorporate them in these 16 lessons, it just doesn't really work.

Constructionist learning theory asserts the importance of giving pupils time to learn partly because it acknowledges that it takes time to create digital artefacts. This has implications for mainstream settings where common allocations for Key Stage 3 ICT lessons are 1 hour a week (38 hours over an academic year). The fact that pupils did not complete their games suggests that game authoring activities need to be carefully

structured and curriculum designers need to consider what constitutes achievable outcomes in limited timescales.

In terms of the pedagogy of game authoring, the data here suggest that skills need to be developed incrementally over the three years of Key Stage 3, and this finding is echoed in similar research in out-of-school settings (see Willett, 2007). If making games is an important context for learning in ICT it needs to be revisited in Years 7, 8 and 9 and software skills have to be taught formally if particular applications are to become creative tools for young people (Willett, 2007: 173). In fact Willett questions whether it is possible for young people to create games because in her research, significant time was taken up with learning software and producing graphics - rather than producing a playable game - to the extent that final projects were hampered by the complicated nature of the software; these findings resonate with experiences in the current research.

5.4 Summary

Just as young people play computer games without direct instruction or reading manuals, this also seems to be the preferred approach for some when making their own computer games.

The extended time frame of the game authoring activity gave pupils an opportunity to engage with a range of strategies for learning, which included exploring the software, using sample games, trial and error, learning from partners, peers and the teacher, using the *Help* menu, following video and print tutorials, accessing the internet, testing and debugging. This finding is echoed in other studies where pupils used a range of strategies to develop the skills they needed (e.g. Baytak et al., 2008; Cheng, 2009).

Whilst pupils valued the constructionist approach in terms of being able to manage their own learning, work in pairs and learn by doing, this way of working was not always efficient. These findings suggest that pupils' enthusiasm for learning by doing has to be balanced by direct, interactive teaching to ensure that skills and features of the software are introduced (see Willett, 2007; Robertson and Howells, 2008) and key programming concepts and game mechanics are modelled. In particular, pupils need access to 'just in time' learning resources, in a range of formats to support them in their individual endeavours.

Chapter 6 Making games – the outcomes

6.1 Introduction

Whilst the previous chapter reported on the process that pupils followed, this chapter looks at the areas of learning that pupils encountered when creating their games, which included creating a game narrative, designing the visual appearance of the game and designing the game play. Chapter 7 extends these findings by discussing what pupils learned about programming concepts and practices.

It is apparent in the following analysis, that the relationship between the design documents themselves and the game pupils went on to make was not well understood and more emphasis needed to be given to carefully completing planning and design documentation and making more use of it throughout the game-making process.

6.2 Creating a narrative

The first activity in the scheme of work was to play and then deconstruct sample games made in *Game Maker*. The purpose of this was to introduce the idea of a game as a constructed system, with identifiable components (theme, characters, objects, settings, goals, sounds, mechanics) and to generate ideas for their own game.

The next task was for pupils to construct a narrative. Game narratives differ from traditional narratives, because interactivity, scoring, game goals and playability also have to be considered. In practice some pupils did not manage to incorporate all these features into their game stories, although they were able to devise scenarios for their games.

6.2.1 Representations

Game narratives generally reflected pupils' interests (horse riding, snowboarding, dodgeball, the supernatural). Where game characters are human, the archetypes of hero/villain (Propp, 1968) and their functions are apparent:

> MD: A girl's father is captured and she goes to rescue her father.

> CB: Our idea is that the main character would be a girl … and she is being chased by the evil woman and she has to run away from her and she sends her evil pigs out to try and (laughs) stop her.
> MH: I think that she should be running back to her house or something, or safety, do you reckon?
>
> MB: In our story … we got a last man … on earth who has to defend the castle from these zombies that were people that took these pills that should have made them immune from cancer but actually turned them into zombies yeah, and then they, like, attack it!

Two of the twelve games feature female characters (girl, princess, witch). Most (5/6) of the boys' games feature male characters (boy, policeman, ghost). Two of the five girls' games include non-human male player characters (Patrick the fish, Starman), and 3/5 girl pairs created games involving animals (cat, mouse, horse, fish), where none of the boys did this. Popular Japanese computer games (Nintendo's '*The Legend of Zelda*' game, *Pokémon*) are also referenced in three boys' games.

Because the target audience for the games they created was 10-11 year olds, most of the game scenarios were fairly benign and challenging the values represented in them was for the majority not required. But two pairs were asked to modify their storylines to avoid negative representations of certain groups, or depictions of gratuitous violence.

For example, one pair (ACJC) had devised a 'burger kid game', in which *'a fat boy [is] trying to gobble up the burgers. There will be vegetables in random places that he has to avoid.'* This pair felt that their initial ideas were compromised by having to take account of the target audience and the need to consider the representations in their game:

> AC: First of all we tried to create a game with a fat boy who has to dodge vegetables and just eat burgers. However, we found out that this would offend larger people so we decided to change our game choice. Then we found out that we had to make the game suitable for a specific age group, that being 7-11 year olds.

In another game, pupils were encouraged to avoid violence. In JDMB's initial ideas *'The last man on earth has to defend a castle from zombies. [The player character] uses weapons that are upgraded every level.'*

> MB: At first we [wanted] to shoot the zombies and kill them but then we found out there was to be no guns or death. So we changed it to throwing a stick or a rock.
> JD: Instead of guns we threw stones. The zombies wouldn't die, they would just be unconscious.

Although this pair modified their game to be less violent, they were reluctant to do so and in the transcript of their voice recordings they express disdain for having to replace guns with stones, bricks and bottles. They had designed a game which involved the player character opening fire as soon as the game was launched, and were frustrated that they could not pursue their initial ideas:

> JD: He was just going to start with a gun.
> CJ: We don't want guns.
> JD: Why not? Lasers?
> CJ: We talked about this before and I said no violence.
> JD: You said no killing humans. We're not killing humans. We are killing zombies.
> CJ: Well try and make it something other than guns.
> JD: Ok, right, shall we have him throwing bricks?
> JD: Or throwing sticks?
> MB: Throwing sticks at zombies?!
> JD: (*Sarcastically*) Yeah, that will knock him out any day!

These examples illustrate that, in creating a narrative for their games, some pupils had to come to terms with the need to reflect on the content, values and representations of the games they created. In doing so, they were introduced to the notion that such factors merit consideration - in the example above a pupil acknowledges that his initial ideas 'would offend larger people' - and this is an important step in young people becoming critical participants in new media culture (Peppler and Kafai, 2007b).

6.2.2 Initial narrative ideas

In general, developing a game storyline raised some unexpected problems. Eight out of twelve pairs referred to difficulties with creating a coherent and convincing game narrative, and incorporating a plausible goal within that narrative:

> KW: You've gotta think why things are there and specifically why they're there and having actually a story that isn't completely impossible, that's believable.

> AC: It was quite difficult to think up [a storyline], especially in pairs, because you've got to have something that's different, that's new, you gotta have a different name and you gotta have something that would appeal to the right age group that you're targeting.

> MH: I think a lot of people found it hard when you said just to come up with a storyline and then people were coming up with storylines but then they weren't suitable for games.

> JG: Finding a good story for a game … was a massive problem for us because we just couldn't find a good enough storyline. We talked and jotted notes down until finally we thought of one.

As in other studies (e.g. Forster, 2006), pupils' difficulties with narrative were evident in the conflicting metaphors and the arbitrary, discordant game characters, storylines and objects they used. Often, their internal mental conception of the game was more sophisticated than the actual realisation of it. Nevertheless, some pairs outlined their initial ideas clearly, chose achievable game mechanics and were able to recreate these in the games they made:

> OWSW: You are a man on a pair of skis and you have to ski down the mountain to reach the bottom. On your travels you encounter objects that get in your way e.g. rocks and trees. To gain points there will be big blue snowflakes that when you ski into them gives you 10 points. There will be a gold snowflake that appears occasionally, this will be worth 20 points. To make them harder to catch they will move. The aim of the game is to get a certain amount of points before you reach the bottom of the mountain to proceed to the next levels. There are 5 levels in total, each with its own different background. As the levels

> increase the speed in which you travel will increase and more obstacles will appear. You will 'wipe out' if you crash into an obstacle. You have 3 lives, if you use all those lives up it will be game over and the game will be restarted and you get a final score. In the end you aim to get a high score.

In KW's 'Shipwreck Escape' maze game the player controls *'Patrick, a fish that got trapped in a ship wreck. [The player] will have to guide Patrick through all the levels avoiding obstacles and collecting pearls. To complete the game s/he must complete all five levels. To complete each level you must collect all the diamonds and pearls and reach the door to progress to the next level. Each level there will be a series of obstacles such as sharks, seaweed, crabs and boulders. At the beginning of the game s/he starts with three lives; when you collide with an obstacle s/he will lose a life. When all the lives are lost the [game is over].'* Because these pupils chose a simple scenario and outlined the main game mechanics they were able to translate those ideas into a playable game.

However, for others this was not the case. Initial ideas were not developed in sufficient detail and did not outline a narrative or identify a specific player character and it is significant that these pupils did not succeed in making a game:

> SARC: In our game we will attempt to aim it towards younger children. The age group will range from 4-6 as we think our game will suit this age group. Our game will be based in a maze, where there are several characters that you can choose to be. You, as the character you chose, will have to make your way through the complicated maze (where there are several complications including water which you can fall in to) to the end. When you get to the end of the maze, there is a pot of money that gives you extra points. Our game is called 'Money Maze'.

This pair later refers to problems they had with their game stemming from not knowing what they wanted to achieve:

> SA: Me and R. weren't very clear from the beginning what we wanted our game to be like ... we just sort of put it together as we went along.

Their strategy of 'putting it together as they went along' was not successful for them. Because they did not have a clear narrative to frame their game, developing the game

play was harder for them to visualise and achieve. This finding contrasts with Papert's view that being guided by the work as it proceeds, rather than staying with a pre-established plan, is a successful strategy (Papert, 1991b: 6).

Other initial ideas were planned in some detail, but pupils did not have the programming skills to realise their game stories:

> LWGW: We are going to try to make a cat and mouse chase. It will be set in a kitchen, where the mouse will be trying to get away from the hungry cat. Cheese will be scattered around. The mouse gets points when it eats the cheese and after eating five pieces of cheese the pace will speed up; this is a new level. On the other hand the cat gets points if it manages to catch the mouse and therefore the cat's speed will also increase. If the cat catches you/the mouse three times you have lost and your score will appear on the screen.

> JBJG: In our game we will have a horse as our main character. The horse will have to fight its way through a forest full of nasty things which she has to avoid, either by jumping over or dodging left or right. In the game it will get harder throughout. Such as level one will be easy with not many things to jump or dodge, then as the game progresses there will be even more things. The things the horse will dodge will be things such as falling branches, animals, logs, hedges and a wolf running towards it. In the game there are also good things for the horse to collect such as apples, carrots and sugar lumps.

In both cases, only some ideas are reflected in the game finally implemented. For these pupils it was necessary to simplify their initial ideas to match their programming skills:

> JB: Our … horse … was meant to dodge things and jump over things but we couldn't get the horse to jump properly, so it doesn't jump, it just dodges the logs.

This was a common problem for pupils in this study - their initial ideas were too ambitious and they were not able to create what they perceived to be interesting game play or sufficient challenge within their games, since to do so required a level of

programming knowledge that they did not possess. This finding is supported in the literature where earlier studies, using different software, found that few students realised the complexities of authoring a computer game or had the programming skills to achieve their initial game designs (Kafai, 1996: 85). This suggests the need to carefully structure introductory game making courses and to give more focus to planning an achievable storyline and the game interactions which arise out of it before implementation begins.

Some pairs struggled with selecting a narratively convincing enemy object for their player character to flee:

> MH: We had trouble deciding on the avoiding object - the object that the character has to avoid otherwise they face the consequence of losing a life. We chose to do a pig at first, but didn't feel this was appropriate so have settled on the idea of spiders.

Others were concerned that the back story for their game was not believable:

> AE: Kokoro lives with her father, Takeshi, who is a mad scientist. One day Kokoro hears her father shout, 'Eureka!' She goes downstairs to the basement, and her father is holding a genetically modified rabbit. He tells her it is programmed to protect her. She decides to call it Saburo. Suddenly a hole is blown in the roof and Saburo runs away. Some 'ninjas' come in. They do not see Kokoro. They say they want the GM rabbit, but Takeshi refuses to tell them where it is. They kidnap him. Kokoro decides that it will be her mission to rescue her father, with Saburo.

This pair later modified their ideas to make the story more narratively convincing to them:

> AE: In the initial plan, the father had created a genetically modified rabbit, which the evil scientists tried to capture. However, we later decided to change this so that he was an archaeologist and had an ancient and valuable scroll, as this was more realistic.

Although this pair simplified their initial ideas, their narrative planning is detailed and specific and this seems to have had a positive impact on the game they finally

implemented, which is the most developed and functional of the group. A strong narrative provided a good basis on which to build their game.

CBMH's initial ideas were also changed in the final implementation:

> Our game will be called 'The Great Escape'. The idea is that the main character will be a girl, represented as a pencil drawing/stick man. Her adventure begins at her house, where she gets a letter saying that her friend has been captured by the evil [witch]. Her aim in the game is to rescue her friend. To get her friend she will have to overcome different stages of difficulty, whilst facing evil pigs on every level.

These initial ideas were changed in the storyboard to *'a princess being chased by an evil witch who is keen to take away her beauty and lock her up. You must escape from the castle at once and find safety.'* Yet in the game later developed there is no castle or witch; the player character negotiates platforms, avoiding spiders and collecting coins. In departing from their initial ideas, the game narrative suffered - there is no discernible, involving goal for the game.

AWTB's initial ideas featured *'an Australian policeman who has to chase a robber through the rundown places in a city. This robber has escaped from jail and has a life sentence. On the way you have to jump over crates and slopes to go up and down. On some slopes there will be oil to help you move faster.'* Establishing a storyline and agreeing on the main player character was a problem for these pupils, who later changed their initial ideas from a maze game to a platform game - which they found difficult to program. Because they did not have a strong narrative, deciding on game mechanics was also problematic. In the final implementation of the game, the narrative identifiers of policeman, robber, crates, oil and cityscape are not realised. The game implemented tells no story and has no goals, because the pupils did not have the graphics or programming skills to reproduce their narrative ideas.

Other pupils (GS, JBLA) used characters from commercial games to help them shape their narratives. GS refers to the *'Pokémon Ranger: Shadows of Almia'* (Nintendo, 2008) computer game and features two Pokémon characters in his initial ideas. However, narrative details are vague - the player controls a *'small person'* who has to defeat *'various enemies and collect items to enhance power and speed. There are five rooms filled with different enemies.'* The lack of a strong storyline may explain why this

pupil did not manage to develop the functionality of his game beyond creating the title screen and programming the directional movement of his player character. Because his initial game story lacked narrative detail, the interactions and game play which arise out of that were harder for him to visualise and implement.

JBLA recreated a *'Legend of Zelda'* (Nintendo, 2004) game and used its protagonist, Link, as their player character, but their final game has only a weak connection to their initial ideas, which centre on *'a man who is driven underground by germ warfare. He is warned beforehand by a mysterious man hiding in the shadows, but it turns out that the man is evil. Then your character is driven underground to a strange vault, which seems safe until strange mutants start attacking. He can't escape because the vault door is locked shut behind him, so he has to go on an underground adventure searching for a way out.'*

The data reported here, which show that pupils had difficulty in creating a satisfying narrative, contrast with the results of previous research, which suggest that creating computer games aids narrative development (e.g. Robertson and Good, 2004; Carbonaro et al., 2005; Robertson and Good, 2006). In these studies the software used, *NeverWinter Nights,* provides characters, objects and terrains and involves branching dialogue. These supporting structures are not available in *Game Maker*, where gameplay is characterised by action, not dialogue. Whilst ready-made game assets can constrain the games pupils are able to make, it seems that some pupils needed the sort of narrative support that these offer.

The data also show that given free choice some pupils may choose unsuitable scenarios, others may choose ideas that are too ambitious or find it hard to create a convincing game story. In order to avoid these potential areas of difficulty, a better model may be to provide game narrative outlines, perhaps based on a social or environmental issue, a model promoted by organisations such as 'Games for Change' (Games for Change, 2013), and 'Apps for Good' (Apps for Good, 2013). Previous constructionist research supported the development of game narratives by asking children to make games to teach younger pupils about fractions (Harel, 1991; Kafai, 1995) or science topics (Baytak and Land, 2011b), and current Key Stage 4 examination specifications offering game-making tasks also use curriculum subjects as a starting point. Although pupils in this research valued being given the freedom to choose their game genre and storyline, they found it difficult to create a game narrative and more support needed to be given in this area. This finding echoes previous studies

using different software, where pupils also experienced difficulties in creating game narratives (e.g. Parsons and Haden, 2007; Baytak and Land, 2011b).

The data suggest that those who did not establish a clear narrative for their games at the start (AWTB, GS, SARC), or did not follow their initial ideas had more problems in creating a successful game. Since the narrative frames the object interactions and gameplay, without a clear narrative it was difficult for these pupils to envision what outputs they wanted in their game and then to implement them.

6.2.3 Game design document

The next stage of the design process asked pupils to complete a game design document. Ten of the twelve pairs attempted this task but analysis indicates that responses were lacking in detail, generalised, or incomplete. For example, when pupils reference the use of sound in their game, they do so only partially: CBMH refer to the background sound for their game as, *'Music'*; JDMB mention *'Bangs and moaning'*; KW writes, *'loss of life - bad sound'*; JBLA refer to *'gun noise, health damage noise, explosion, gain health noise'*. In general, pupils did not list all the sounds in their game, e.g. they omitted feedback sounds or sound effects. ACJC, AWTB, GS did not complete this section of the game design document.

Pupils likewise did not give a complete account of the gameplay. JDMB's player character *'Starts on walkway, object in hand ready to throw at zombies moving towards you'*. There is no mention of rewards/penalties or win/lose states. In describing the levels in their game, some pupils did not give a clear overview of level design or progression across levels. JBJG state '*3 levels get harder each time*' but there are no details of how the challenge increases. JBLA refer to one level '*but it is really big with different areas.*' CBMH, ACJC, AWTB, GS did not complete this section.

These omissions suggest that pupils found visualising the detail in their games difficult or were reluctant to have to plan the game on paper, preferring to implement it directly in the software. Indeed, computer-based activities took precedence over the written planning documents and a significant amount of time was taken up at this stage in locating or creating and editing graphics for their game. Nevertheless, in missing out important details at the planning stage, pupils encountered problems later.

This section has summarised observations arising from an analysis of the design documents which pupils produced at the beginning of the project. The next section

looks at the areas of learning pupils engaged with as they began to design the visual appearance of their games.

6.3 Designing the visual appearance of the game

6.3.1 Storyboard

As part of designing the visual appearance of their games pupils were asked to represent their game as a storyboard (see Appendix 8). Seven of the twelve pairs completed this task. Interestingly, there were several significant omissions in terms of graphical information in the storyboards pupils submitted and these are detailed below.

KW represented her game across six coloured frames, illustrating the rooms, and her player character but omitting the game collectables. The storyboard has only a weak connection with her initial ideas and the appearance of the maze game she later made.

AWTB's storyboard lacks an image or title on the title screen and represents the game as a series of black and white mazes. It has only a weak connection to the final platform game implemented. The storyboard indicates lives/score and timer mechanics and game challenge, but does not convey a strong visual sense of what the game should look like. In particular, there is no representation of the player character or other game objects. It is significant that in their final game, the title screen is an abstract swirl of colour and does not relate to the game narrative; other game graphics (backgrounds and sprites) do not reflect their initial ideas and are not developed (the background is grey, the player character is an arbitrary shape). For these pupils the absence of graphical information suggests that they were not able to visualise their game and this is carried through into the final implementation.

SW's storyboard illustrates the title screen and represents the flow of the game in six frames. She labels the player character and other game objects and includes the lives/score and 'game over' screen. She also labels elements of game play ('Character collides with snowflake to gain points'). Her partner, OW presents a black and white drawing of the player character, title screen, main game screen and 'game over' screen and labels game objects, score and level. These stronger details enabled the pair to create a more successful game. Because they were able to visualise what they wanted their game to look like, it was easier for them to recreate their ideas.

LWGW's title screen includes the title and menu buttons but lacks any images. It seems that these pupils interpreted the instruction to include a title screen literally, giving only a title - and this was the same for 5/7 storyboards completed. The storyboard illustrates and labels the player character and other game objects and hints at the background, but this only has a weak connection to the maze game they finally implemented.

MH's black and white storyboard depicts a title screen which lacks a background and navigation buttons. The player character is indicated, along with another character, which does not appear in the game later implemented. A back story screen and three levels illustrate the platforms and game objects; lives/score/level information is also shown. The 'game over' screen depicts a high score table and labels the condition for the high score table to display ('if you lose all your lives'). Level progression is labelled and a key to the game objects and keyboard symbols (left, right, up, down arrows and space bar) is given. Her partner, CB indicates a black and white title screen with title text and 'Click to start game', but there are no graphics. A menu screen displays four buttons but gives no other visual information. The game room is shown and includes level/health bar, player character and other game objects. A 'how to play' screen gives game instructions; a 'high score table screen' references the score mechanic but there is no button to navigate away from the screen. A 'game over' screen includes text but gives no other visual information and no exit button.

In ACJC's black and white storyboard, a 'game start' screen displays the start button and controls and includes game instructions but lacks a title or graphics. The level 1 screen shows player character and obstacle object but no reward object, or score/lives mechanic. A net is depicted but no other background features are shown. The level 2 screen adds an image of the reward object. A high score table, menu and credits are indicated but no other visual information for these screens is given.

AEMD's storyboard displays four levels but their final game only makes use of designs for levels 2 and 3. No title screen or instructions are referred to, although these were later implemented. Level 1 indicates the player character and other game objects, but there is no reference to score/health/lives mechanics beyond bonus points. Labels hint at the storyline but no details of game play or colour information are given. Levels 2 and 3 show the player character and obstacles and one feature of the background is depicted (clouds). Levels 3 and 4 label some elements of gameplay ('level restarts if player runs out of fuel', 'jump on henchmen to destroy them').

In summary, the storyboards omit some key features of the games, notably colour, player character, obstacle and reward objects, score, background graphics, title screen and interaction buttons. This is surprising given that the games pupils play are generally graphic rich and visually appealing. It seems that at this stage in the design process pupils had difficulty in visualising the game in its entirety and in representing its separate components graphically. It may be that pupils struggled to represent interactive media and moving imagery in two-dimensional paper format. These findings suggest that pupils found it difficult to create abstractions of their games, such as the storyboard and the design document, and found concrete production in the software more accessible and appealing. Moreover, pupils did not make much use of these planning documents to support them in making their games and more emphasis needed to be given to the purpose of planning documents in the game authoring process.

6.3.2 Graphics

Once they had completed the planning tasks, the next stage was to produce the graphics for their game. Most pupils used *Fireworks* (Macromedia, 2004) to create or edit their game graphics, while others used ready-made graphics, sourced either from the internet or from *Game Maker's* sprite library. Creating game graphics (sprites, backgrounds, title screens) was a new area of learning for these pupils and presented them with many challenges, not least of which was locating suitable items:

> JD: It was quite hard looking for sprites, like, um, if you typed in something [in the web browser search bar] it might not always come up, 'cos of the [LEA] ban.

> KW: One of my first problems was I was trying to get a bad sprite, like something to be the bad guy, but I couldn't find one on the internet and then when I found one it wouldn't import and I couldn't use the resources that were already in *Game Maker* for some reason.

> JB: At first we were gonna have the screen scrolling downwards, and then the only [animated] horse we could find was going ... [horizontally].

One pair thought that the software should have given more support for graphics:

> AC: I think the graphics could do with a bit of improvement. 'Cos basically it's your sprites, all your stuff, you don't have like, the rooms don't have anything, it's all yours.

This pupil articulates the demands of having to create from a blank canvas. Although pupils had access to a limited range of backgrounds and sprites within the software, these were not suitable for their individual storylines. They needed access to a greater range of ready-made graphic resources, such as those provided by other software used to create games, e.g. *Scratch,* or in online tutorials (e.g. Aardman/Nominet Trust, 2014). Creating and editing graphics dominated the early sessions and this was exacerbated by the fact that pupils' had limited exposure to graphics software and lacked image editing skills. While it was important to give pupils the opportunity to create their own game graphics, because they enjoyed doing so, in practice it was time consuming to create satisfying graphics and the outcomes were often disappointing - 7/12 games included poorly executed or very simple sprites (see Appendix 1).

Pupils also encountered difficulties with creating successful background images, which were not well drawn in 5/12 games; others found it hard to visualise their backgrounds:

> TB: Also [a problem is] making good scenery that will keep them interested. A lot of good scenery, it's quite hard to keep thinking of new ideas for scenery.

> OW: We were going to have a different image as the background on each level but then it became too hard as you can't really have different images of a snowy background!

Overall, pupils spent too long locating, creating or modifying graphics, which meant that there was less time for programming the game action. This finding is echoed in other studies, where the process of making game graphics became the overwhelming focus, pupils spent more time on interface design than program logic, and combining images from different software complicated the production process (Kafai et al., 1997; Shackleton et al., 1997; Lin et al., 2005; Parsons and Haden, 2007; Willett, 2007; Northcott and Miliszewska, 2008; Baytak et al., 2011; Smith and Sullivan, 2012) and was conceptually and practically challenging for those with little prior experience (Macklin and Sharp, 2012).

Notwithstanding these problems, creating graphics for their games introduced pupils to many new concepts and skills and some expressed satisfaction with this component of their work:

> MD: I liked making sprites, that was fun using *Fireworks*. It was useful and deciding on the sizes we needed for them and stuff like that.
>
> CB: I like the graphics in our game.

Modifying graphics they had created themselves was sometimes problematic. Five out of twelve pairs experienced problems in resizing their sprites appropriately and 3/12 games included oversized images (JBJG, JDMB, SARC):

> AE: We had to make [the sprites] quite small and so we had to learn how to make them so that you could recognise what it was, but only using 16 by 16 pixels, which was quite hard.
>
> SA: Well we made our sprites too big and we had to change the size of them, which was a bit annoying.
>
> MD: I had already made these characters and they were 32 by 32 and [then] I had to shrink them but it just degraded the quality loads.

The notion that the canvas size must be the same as the image size of a sprite was not intuitive:

> JC: We had a bit of difficulty … with getting the pictures you created on *Fireworks* to fit into the [grid] and you got an awful lot of wasted space on your sprite so you got like … in *Fireworks* you created this little sprite and then you'd have loads of white background.

In some games (JDMB, SARC, JBJG) where sprites had a large surrounding canvas, this prevented the object from being correctly placed in the room and made collision detection inaccurate. Sourcing images from the internet introduced further problems:

> AC: We had a bit of trouble because we incorporated a Mars bar in our game and we found a picture of them on a website but it had a shadow underneath it,

> so when we came to creating the sprite, when we put it into the room it had this black outline underneath and on the background it had white edges.

Pupils had to learn to make sprite backgrounds transparent. One third of pairs (ACJC, AWTB, GS, JBJG) found this difficult to achieve or included graphics with non-transparent backgrounds (GS, JBJG).

Beyond these specific skills, pupils also learned about the visual conventions of two-dimensional computer games (e.g. vertically or horizontally scrolling backgrounds, points of view, animation) and they used visual skills when they made aesthetic decisions about colour, texture, size/scale, composition and perspective. The 2D games that pupils created were predominantly graphical; text was used only in title screens, messages and game instructions. In this respect making a game was different from previous units of work, which, while they may share a visual dimension, normally include significant textual or numerical content.

In designing the visual appearance of their games pupils also developed greater understanding of how to conceptualise 'screen space'. Although pupils in this study had previously created screen-based systems, such as web sites, multimedia presentations, or animations, creating a 2D computer game involved learning to view screen space in a different way. The screen in a game context is a Cartesian plane, a space mapped by coordinates and measured in pixels and they had to learn how to define and manage the position of objects located within this space.

They learned that to control the 'layering' of elements on the screen a value for 'depth' must be specified, to determine whether an object lies in front of or behind other objects. Two of the twelve pairs used this feature to control the layering of objects in their games (JBLA AEMD). However, in four games (JBJG, JBLA, LWGW, AEMD) depth was not correctly configured for some objects, which affected their display.

Although the development of graphics skills is important in its own right, the findings of this study suggest the need to carefully consider the place of graphics instruction within game authoring activities. Time allocated to sourcing, creating or editing graphics needs to be limited, and the choice of software used needs to be carefully considered. In this study, the graphics editing software selected had a learning curve of its own and gave little support to novice users. This problem of using professional standard

software is also described in other research related to digital media production (see Willett, 2007).

To reduce the time spent on learning graphics skills it may be preferable to use the image editor in *Game Maker* to create bitmap graphics, using more straightforward 'pixel art' techniques, or to make greater use of ready-made sprites. This allows more focus to be given to the programming task and removes the need to learn additional software skills. Alternatively, if such software is to become a creative tool for pupils, skills need to be developed incrementally over time with increased exposure to the software and practised over the key stage (Willett, 2007; Robertson, 2012).

6.4 Designing the game play

Even though the games the pupils made were not sophisticated, the design of meaningful game play involved complex thinking - pupils had to consider the relationship between player action and system outcome and to make outcomes clear to the player in the form of visual and/or audio feedback. As players of games pupils take this for granted; when making their own games they had to think about these dynamics explicitly, perhaps for the first time. This section illustrates some of the elements they had to consider when designing the game play.

6.4.1 Animation

Five of the twelve games included one or more animated graphics (AEMD, JBLA, JBJG, LWGW, JDMB) either sourced from the internet or created by themselves. Pupils felt a sense of achievement when they managed to include this feature:

> JBr: I've learned that you can make games more advanced than I thought you could, like the idea of being able to use more than one sprite as an object so you can get the effect of moving.
>
> MD: [The animation] was fun. I hadn't really done something like that before. I didn't know you could use *Fireworks* for animation.
>
> JB: Yeah, I'm glad that I managed to get the horse to move.

One pupil (AE) produced an animated splash screen for his game, in which a light bulb flickers to illuminate the title (see Figure 14). He learned to control the animation speed

and duration by defining the number of frames required and the number of frames per second.

Figure 14: AE's animated splash screen

Pupils learned how to import animated graphics they had created in *Fireworks* or located on the internet into *Game Maker*, though this was not without problems:

> AE: We've made an animation for the intro but when we insert it into *Game Maker* it changes the frame rates so we are trying to get around this problem by changing the animation length.

> JB: We had a problem with the horse. When we first got it, it moved really fast and then we managed to slow the horse down.

In three games (JBLA, LWGW, JBJG) an animated character played too fast. Pupils learned that the rate at which the animation plays depends on the room speed setting. In another game (AEMD) pupils had to make sure that an explosion animation travelled downwards at a convincing speed so that it appeared to follow the graphic of a car. Other difficulties occurred in specifying the correct position of an animation (JBLA) and sequencing animations so that one animation ran its course and then disappeared before another animation played.

In 3/12 games (AEMD, JBLA and GS) multiple sprites were used to modify an object's appearance when it changed direction, and this feature was particularly noticeable when it was lacking, as in KW's and LWGW's games where a fish appears to swim backwards and a mouse reverses.

Seven of the twelve games did not include animated sprites, yet it is not difficult to

achieve simple animation or to change a sprite's appearance when it changes direction. These are core game functionalities which pupils need to be taught, if their game graphics are to be successful.

6.4.2 Usability

Although the importance of considering audience was explicit in the National Curriculum framework for ICT in operation until 2012, that audience was generic, implied, possibly not authentic and normally invoked to ensure the appropriate use of font style, colour, image, sound and similar features. In contrast, the audience for a game interacts directly with the end product and this leads pupils to consider features beyond presentation, such as user experience:

> AE: You are having to think about lots of different things that could happen.
> MD: [It] makes you think. [It] makes you more aware of how people think.
> AE: Yeah and you have to imagine all the things that people might …
> MD: You're the player …
> AE: Yeah [you have to] imagine you're the player.

> MD: I like the high score table. That's a good thing to have because it makes [the game] more competitive.

> AE: Also the variety of different levels makes it more interesting for the player, because rather than just playing one thing, they won't get bored.

Computer game authoring gave pupils an authentic opportunity to design for usability and this is evidenced in their games when they created title screens, wrote game instructions, made use of common control options (e.g. arrow keys for directional movement), gave the player feedback (score, sound, text), and made choices in terms of interface design (theme, character appearance, animation, level design). The journal extract below illustrates design decisions made by one pair (AEMD) to enhance the usability of their game:

> At first we couldn't decide whether to make the spacebar start the game from the title screen, or make a mouse click on a button to start it instead. We decided that as all of the controls in our game use the keyboard, we would stick to this throughout the game. We had seen several other games which used a key press to start the game too.

We decided that in level 2, the scrolling car would only go left and right and not up and down like in level 1. The player just has to dodge the obstacles in the game, and we thought that adding another direction for the car to travel in would confuse our audience.

We will also add another room to our game which will congratulate the player on completing the game. It will display a 'Well done' message. The player can then exit the game using the Esc key, or play again by pressing the F12 key. They can also enter their name if they have achieved a high score.

As the game is quite long, we think that we should have quite a good reward when the game is complete. As well as a 'Well done' message, we could also display a short video/animation. We considered changing the global game settings so that the player cannot exit the game or log off until they have completed the game. However, this would annoy and anger most players so we probably will not do this.

Other aspects of usability are concerned with how the game 'communicates' with the player. For example, AEMD customised their game by designing a graphic to display in the title bar, as well as creating graphics to indicate the loading of the game and to display the player's lives, health and score status (see Figures 15, 16 and 17).

Figure 15: Title bar game icon

Figure 16: Loading graphic

Figure 17: Score, health and lives status bar

Pupils also showed awareness of designing for usability when they created interaction buttons with rollover effects (GS) or designed screens which included user options beyond starting the game (GS, AEMD, KW, OWSW), as illustrated in Figure 18.

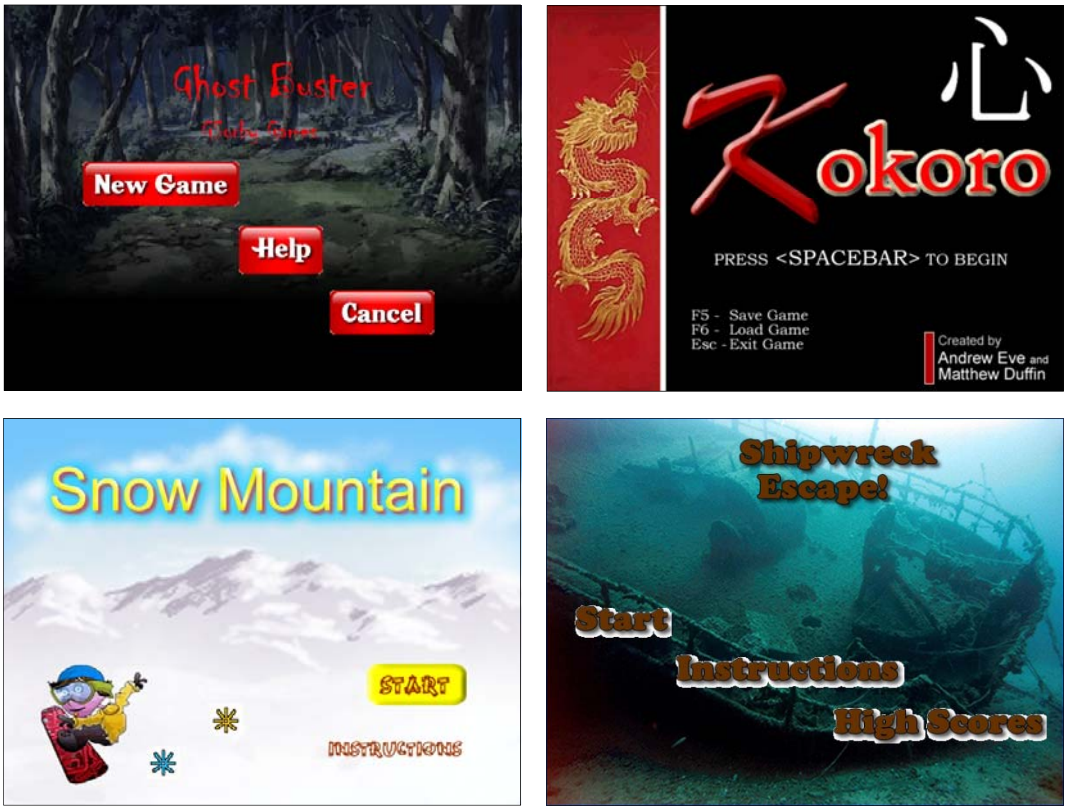


Figure 18: Title screens offering user options

Further aspects of designing for usability are evident in the 3/12 games which include instructions and the two games which include a high score table (see Figure 19).

Figure 19: OWSW's instructions screen and KW's high score table

Three games used messages to communicate with the player (see Figure 20). JBLA used messages to instruct the player how to start the game, and within the game, to

add dialogue ('Help me!', 'Ha ha you fool, I'm free!'), to advise the player of a game state (a door is locked), and to invite the player to interact with the game ('If you touch me you get a wish'). ACJC made use of messages to advise the player that they had lost all their lives, and to congratulate the player on winning the game ('You are the ultimate dodgeball champion! Well done dude!'). AEMD included messages to support the narrative of the game and to inform the player that they had completed a level or finished the game ('Well done! You have successfully flown to Tokyo and completed level 1!').

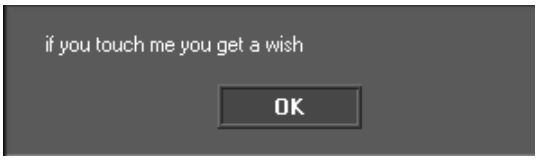


Figure 20: Use of messages

In incorporating these elements, pupils show a developing awareness of aspects of usability as an important feature of digital media production.

6.4.3 Interactivity

For the pupils in this study, designing and creating the interactivity in their games was a new experience. Although they had previously created interactive elements such as clickable buttons in web sites, presentations, spreadsheet systems, and database forms, they had no prior learning of creating the sort of event-driven interactivity involved in a computer game. In a game, every input has an output - something happens - and this has to be designed and implemented. Pupils grappled with the complexity of creating the multiple interactions in their games:

> AE: You have to think of, like, all the possible things that could happen … and all the, like, rules. Like, with the *Flowol* thingy, only one thing can happen, or, say you have an input … or only two things, but with [*Game Maker*] the player can move anywhere on the screen so … you are having to think about lots of different things that could happen.
>
> JG: I liked it but I think it made you think too much … so, like, when you had to have one object do that, and then you had to think, oh wait, but that one has to do that and the other one ... and it's just a bit confusing. Like, thinking when the

> horse had to, like, touch the log and then the log has to disappear, but the horse has to stay there and then you have to let the horse touch the apple …
> JB: And make the apple disappear.
> JG: Yeah and then make them reappear and it's just really confusing … But it's fun to make, just … all muddled up.

Pupils refer here to the demands of creating an open system, as compared to the closed control systems they had previously created when engaged in programming activities, which had pre-defined purposes and finite inputs and outputs. Whilst this progression in complexity is important for learning, the data here suggest that some pupils needed more support, perhaps in the form of templates or partially completed games which they could modify, so that they are guided in how to create the common interactions and mechanics in games, before they construct their own.

The 'confusion' referred to may also be due to the working approach taken by those pupils. Their preferred style was 'bottom up' bricolage, rather than 'top down', systematic and planned (Turkle and Papert, 1990). The data here suggest that, in terms of creating the game interactions, pupils needed to produce more detailed, precise and systematic plans, before they began to build their games. Creating the interactivity in a game requires pupils to decompose the game action into multiple separate units and this introduces them to the idea that programs are 'modular', but pupils needed more guidance to adopt these approaches, which did not come naturally to most:

> AC: We naturally wouldn't have thought of that. If you asked us to create a game we would probably just say, arrow keys move forward, if you get this add 10 points, not 'where does the thing that you collide with go?', not 'if you release the key will it carry on moving?' or stuff like that.

Envisioning and creating interactivity was a new way of thinking for pupils, and they recognised that this set the work apart from other types of work they had encountered. This was an important area of learning for them because they saw that they could make things happen and understand the basic principles involved, and this dispels the notion that only professional developers can create software (Papert, 1993; Noss et al., 2012).

6.4.4 Sound

Although all pupils referred to background sound and/or sound effects in their planning documents, few managed to implement this feature successfully. Five of the twelve pairs made some attempt to include sound in their games; of these, three managed to do so effectively. One of the five games made by girls included a sound file in its resources although it was not implemented. Four of the six games made by boys included some attempt to implement sound.

Some of the problems pupils encountered were due to simple errors. For example, ACJC loaded a sound file and included a *play sound* action in the *create* event of their player character, however they selected the wrong sound file so this did not function. In another case (JBLA), a sound file does not play as intended since it is programmed to play in the *create* event of a collectable object, so plays once when instances of the object appear on the screen, rather than each time the object is collected.

Others did not have time to implement sound across all levels. AEMD added background sound to the splash screen, title screen and level 1 of their game but not in level 2. Some sound effects are implemented in levels 1 and 3, but others are noticeably absent. For example, when the player's plane fires missiles, no sound accompanies this action, yet other interactions on this level are accompanied by distinctive sound playback. GS loaded three sound files in his game but only one sound plays.

Controlling the playback of sound to synchronise with the game was also problematic - in one game a sound file loops even though looping is set to 'false', or sound continues to play after the level has ended. In another example, (JBLA) background sound plays on game start but is followed by an extended pause before the sound file restarts.

In fact, programming a sound to play in *Game Maker* is straightforward - and the reason that pupils did not implement sound effectively appeared to be because they did not have time or because this aspect of game design was not perceived to be as important as developing the graphics and game play.

6.4.5 Timing

Another factor that pupils had to consider when designing the game play is that games have a temporal dimension. Constructing a computer game involved making things happen at certain intervals in the time frame of the game, introducing delays in the

game play, or determining the moment in time and the frequency that a particular action should occur. They learned that units of time in *Game Maker* are measured in frames, steps and milliseconds.

Pupils learned that a *step* is a short period of time - 1/30th of a second - and that a *step* event can be used to control the frequency of actions which recur throughout the game. For example, AEMD used this event to control the timing of the firing of bullets:

> AE: First we need to test ... whether they've already fired a bullet in the last 15 steps.

These pupils also learned to manage timing by using the *alarm* event, which allows things to happen from time to time in a game. The *alarm* event was used successfully to control the timing of their splash screen, so that it appeared for a duration of 4 seconds and then transitioned to the game start screen; it was also used to control the interval at which missiles could be fired, (every 15 steps, rather than continuously). The *sleep* action was used in the same game to pause the game action after an explosion animation had run its course, signifying that the player's plane had been hit, and to delay the appearance of the winning message after the game had been won.

The use of these timing events and actions in their games shows awareness that, in terms of game design, timing can be controlled to enhance the game play experience.

6.4.6 Challenge

In designing the game play, pupils also had to think about how to create the right level of challenge and this was not always easy to achieve:

> LW: We struggled on thinking how the game could get harder; we thought maybe [increase] speed or amount of cats. In the end we decided to increase the number of cats.

> GW: Problems we had with our design were that it was possibly too simple for older audiences.

> JG: Deciding on the game rules was very hard because we had to keep it easy, but not too easy and hard, but not too hard.

In games which featured more than one level (ACJC, AEMD), challenge did not significantly vary between levels. Generally, attempts to create challenge were characterised by making enemy/obstacle objects move with increasing speed, or by increasing their number. Sometimes challenge was set too high, as in AEMD's level 1, where enemy bullets fire too fast, too often, or ACJC's level 2 'Dodgeball' game, where too many balls fall too fast and not enough rewards are generated. In games which included one level, challenge was minimal - for example, KW's maze game features no obstacles and no mechanism to pass to level 2; in some games (GS, SARC), there are so few interactions possible that challenge is non-existent.

In many games, the planned challenge could not be realised because the programming of the game was incorrect or incomplete. For example in AWTB's platform game the challenge of manoeuvring the player character to the highest level is compromised by the fact that its jumping mechanic is not well implemented. LWGW had planned increasing challenge in their 3 level maze game, but this could not be realised because the programming of their game objects was not sufficiently functional and there was no mechanism to progress through the levels. JBJG's game lacks challenge because obstacles and rewards are not fully implemented, so there are limited interactions and no mechanism for scoring points. The challenge in JBLA's game is compromised because the score mechanic is not correctly implemented.

It becomes clear that a wide range of complex design tasks are involved in creating a computer game (narrative, graphics, animation, usability, interactivity, sound, timing, challenge) and it is not surprising that pupils encountered problems, since they had no prior learning of many of these areas. A key theme in this section is that across several areas (narrative, planning the visual appearance of the game, designing the game play) pupils found it difficult to visualise or conceptualise their games in the level of detail required for the planning tasks they completed to be useful to them. Pupils found it difficult working with a 'blank canvas' and needed more support than the examples provided. This has implications for the pedagogy of game authoring at this level. Whilst the findings in this section show that pupils gained an awareness of the main areas involved in game design and some experience in developing aspects of these, it seems that such an open-ended task was too demanding in terms of the range of skills pupils needed to learn in the time available. To reduce these demands introductory game authoring courses should provide ready-made game assets (sounds, graphics, backgrounds) and narrative outlines, so that pupils can focus on

learning how to program the game interactions rather than directing too much attention to the game storyline and aesthetics.

6.5 Talking like game designers

As they made their games, pupils began to adopt a game designer discourse. They enjoyed using new words for the new concepts they encountered and applying known words in a new context - and some (AEMD, ACJC, OWSW, KW, JBLA) became quite fluent in this language.

In common with other studies (e.g. Games, 2010), as pupils began to appropriate the language of game design, they increased their understanding of how games are constructed and developed 'production-oriented' technology-associated literacies (Salen, 2007). Using *Game Maker's* actions introduced them to the mechanics of game design (actions: *move, jump, create/destroy instance, play sound, next room, set lives/score/health*; events: *collision, key press/release; step, alarm, mouse*). They learned about game components (*sprite, object, instance, action, room*) and also began to use more abstract words to describe states, behaviours and interactions of objects (*solid, visible, collision*) and to refer to programming concepts (*event, input, output, repeat, test/check variable*).

This language learning is important because once they became even a little fluent in it they became able to 'speak' things they would not have been able to articulate previously. The data here support Papert's observation that children can appropriate terminology and concepts designed to articulate the process when they want to make the computer do things, and in so doing they become more articulate in developing formal systems (Papert, 1980c: 162).

Some pupils expressed a feeling that learning to use this language enabled them to participate in conversations with others more knowledgeable than themselves:

> CB: I had no idea about sprites and objects and rooms.
> MH: Yes, now I feel I could talk to someone really …
> SA: I feel I could talk to somebody who knew more about it, like J. or A. for example.

In their journals, some pupils displayed a developing ability to articulate the process of making a game in *Game Maker*:

> MD: Another event needs to be created for the movement of the car. There is a *key press* event which can be assigned to many different keys. For example, the 'up' arrow. Then you need an action. Normally you would just set a movement action which would make the car travel forward, but in this game there is no wall to stop the car, so we need to use variables. A variable constantly asks a question and when it is true, allows an action to be performed. For the *left key press* event we place a variable, 'If *x* is larger than 40'. Then we can tell the car to move relative to -4 on the *x* axis (this moves the car left).
>
> All the enemy cars use the same actions and events, except the scrolling speed is slower to give the effect that they are also moving, but slower than the [player] car. The *jump to given position* function is set to *x* = random (room_width) and *y* = -50. This means that the cars appear in random positions above the screen. This eliminates the look of repetition that games can sometimes have.
>
> I needed to apply actions and events to give the illusion that the car was moving. To do this I simply made a *create* event and told the object to start moving down at a speed of 5. Then, to make the [white lines in the middle of the road] carry on scrolling constantly I created a variable which said, 'If *y* is larger than room height', then I told it to jump to a given position which was *x* 300 and *y* -48.

In this extract, the pupil refers to using a conditional statement to test a variable (the *x* coordinate of an object), although he uses the word 'variable' for 'condition'. His use of language reflects his emerging understanding of how to construct game programs.

6.6 Use of software

In the course of the game making activity, pupils used two programs they had not encountered before - *Game Maker 7* and *Fireworks 2004 MX*. The ICT curriculum has been criticised for only teaching skills in the use of office productivity software (see

Furber, 2012; Gove, 2012b), but in this project, pupils increased their technological fluency and learned to use new software for new purposes.

Using *Game Maker* developed pupils' digital literacy in so far as its interface and operational features differed from the software commonly used (see Figure 21). *Game Maker* is not a 'WYSIWYG' ('what you see is what you get') environment i.e. its display does not precisely represent the appearance of the game. As an integrated development environment (IDE) it includes a programming environment, an image editor and a compiler. The game functionality is constructed in the programming environment, and is then compiled by the software at runtime, at which point the game is rendered in its visible playable format. Pupils were continuously switching between the abstract programming and concrete execution environments and this was a new experience for them:

> LW: I have learnt a considerable amount in this project. *Game Maker 7* was some new software and completely different to others that I had used before so I had no knowledge of the software. Also I had only used *Fireworks* once. I found both *Game Maker* and *Fireworks* difficult as they have more to offer which makes it more complicated and it is laid out differently.

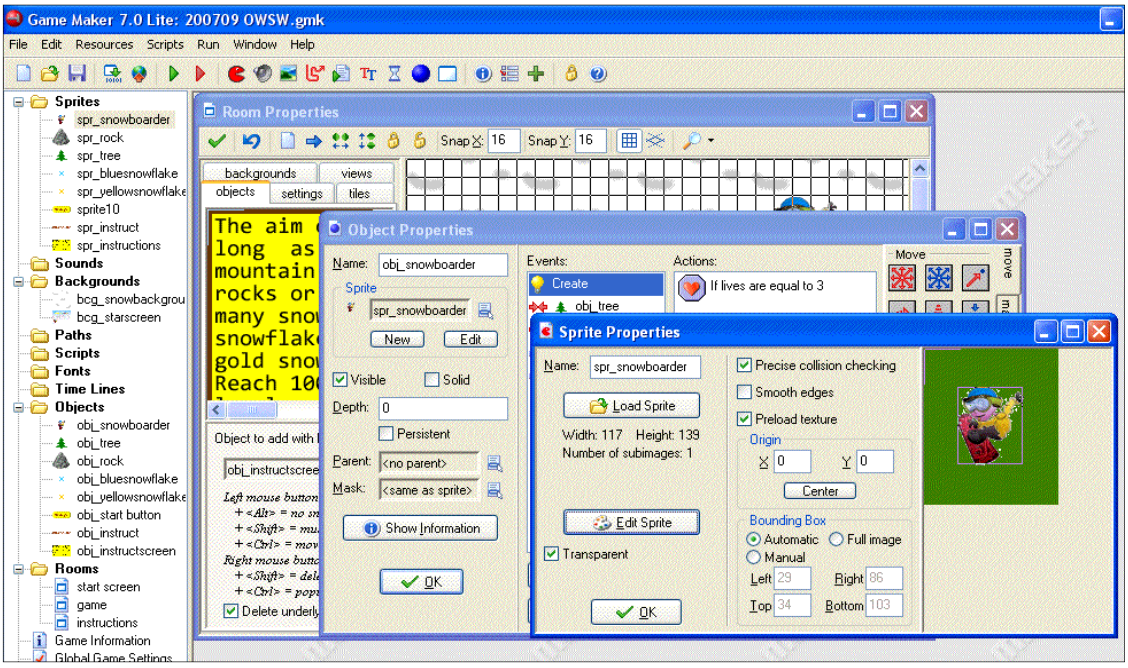


Figure 21: Game Maker's sprite, object and room properties boxes

Some pupils (AEMD, JBJG, JBLA, JDMB, LWGW) made use of the sprite editor, another component of the IDE. In these respects pupils were working 'at one remove'

from their game. This is different from working with a word processing or desktop publishing document, a spreadsheet or a graphics program, where there is no need for the compilation step.

The process of creating a game in *Game Maker* is also different from the types of activity involved in using office productivity software. Pupils create sprites and background graphics and load them into the game. Game objects are then created and assigned a sprite to give them a visual appearance. Each object is programmed with events and actions which determine how it functions within the game. Actions are set by dragging icons into the actions panel and properties, settings and parameters are applied to them. Rooms are created, which constitute the levels in the game, assigned a background appearance and configured to determine their size and whether they scroll or not, for example. Objects are placed within the room. The game can then be run and the game action viewed on screen.

This way of working emphasises the 'constructedness' of a game and encourages pupils to view digital media as modular systems. It prioritises functionality rather than presentation as the dominant outcome of their work. It also introduces the idea that the game's visual appearance is separate from the underlying program behaviour.

6.7 Summary

This chapter has considered the areas of learning which pupils encountered in terms of game design (constructing the narrative, visual appearance and usability of the game) and has shed light on some of the difficulties they encountered. Pupils' achievement in these areas as they are evidenced in the games they created was evaluated using an adaptation of the SOLO taxonomy (see Chapter 4) and is summarised in Appendix 1.

In terms of developing a pedagogy of game authoring, which identifies what concepts pupils find difficult, the misconceptions they may hold and how to address this (Mishra and Koehler, 2006: 1027) it emerges from the findings here that planning of the game narrative, visual appearance and interactions is an important part of the process and while pupils may resist planning tasks, preferring to learn by making their games, it is not productive for them to do so. In particular there is a need to focus on tasks which support pupils' understanding of game programs as modular constructs, composed of separate entities (see sections 7.2.3 and 8.3.2). Additionally, pupils need to learn to be

systematic and encouraged to visualise and represent the significant graphical and functional detail in a game, before they begin to implement it.

The next chapter considers what pupils learned about programming concepts and practices as they made their games.

Chapter 7 Learning to program with *Game Maker*

7.1 Introduction

This chapter documents the programming concepts and practices evidenced in the data collected, drawn from pupils' authored games, planning documents and interviews, and considers to what extent using *Game Maker's* visual environment supports pupils in learning basic programming concepts. Throughout the chapter, examples of programming code are identified with pupils' initials and presented in a textual format which corresponds to the code created by pupils using graphical symbols (see Figure 23). A discussion of the difficulties encountered follows in Chapter 8.

In this domain specific study the term 'programming' is defined as "the act of assembling a set of symbols representing computational actions … [to] express intentions to the computer" (Kelleher and Pausch, 2005: 83-84). In learning to program, pupils are introduced to three key processes - sequence, selection and repetition.

Programming in *Game Maker* requires pupils to create sequences of events (inputs) and actions (outputs), which define the performance of elements in the game. This chapter considers the learning that pupils achieved with this approach, using a symbol-based, drag and drop environment. I suggest that making computer games is a suitable pedagogical model for learning basic programming concepts, since domain-specific programming, (in this case, computer game authoring), is more accessible for novice programmers (Smith, 2000) and can make learning about programming more concrete and motivating.

The pupils in this study had some prior exposure to basic programming when they used software to construct flowcharts to control on-screen simulations of systems, such as a theme park water ride and a Ferris wheel. This introduced the concepts of input/output, loops, decisions, sub-routines and variables. They used these constructs to control closed systems which featured a finite number of inputs and outputs. The game authoring activity developed their understanding of programming, since a game is a more open system and involves defining a wider range of inputs and outputs and their parameters.

7.2 Learning to program

Transcripts of pupils' voice recordings, written documents and interviews were coded for references to programming concepts (see Appendix 5). The program code pupils used to construct their games was categorised and analysed according to the programming concepts listed in Chapter 4 (see Table 2).

Figure 22 below illustrates how *Game Maker's* visual environment represents some of these programming concepts.

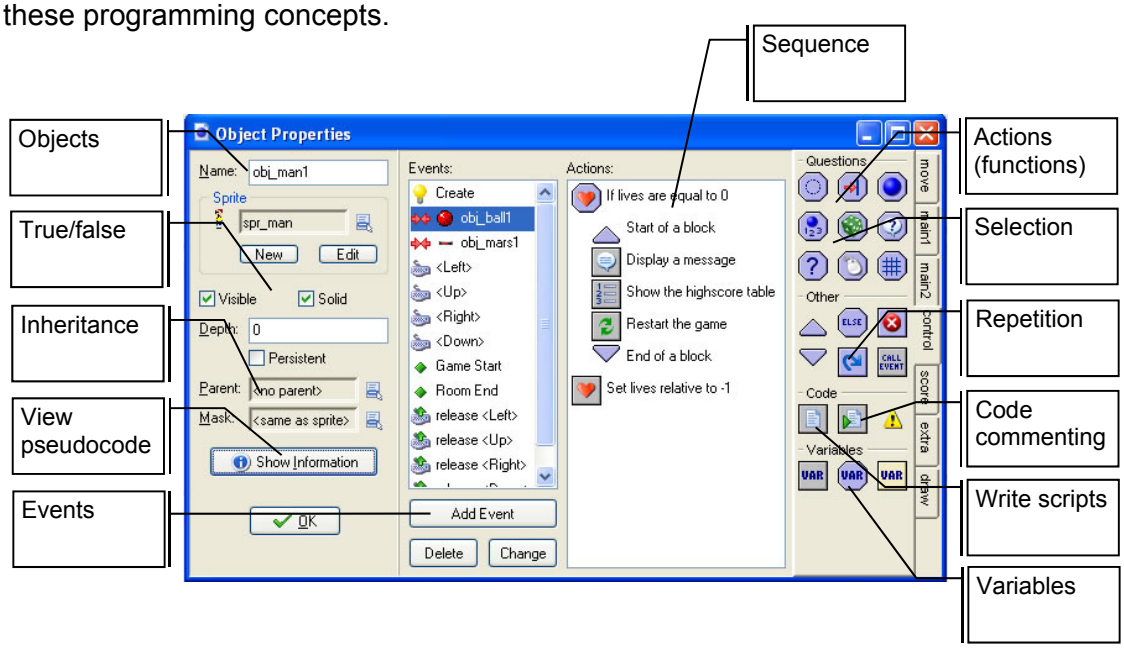


Figure 22: Programming constructs in Game Maker

7.2.1 Sequence

The concept of sequence is important in designing and writing computer programs (CAS, 2012a). Creating a game in *Game Maker* involves selecting events and actions for an object and putting them in a logical order, since they are executed sequentially from the top, downwards. Pupils learned that the sequence in which they order events and actions has an effect on the order in which events and actions occur in the game:

> TB: You have to think about … the input and the output all the time ... which order the programs go in, where they should go, what they should be on ...

In this respect using *Game Maker* supports the development of algorithmic thinking, whereby pupils learn to define specific instructions for carrying out a process, in a visual format. The visual algorithm can also be viewed in textual format (see Figure 23).

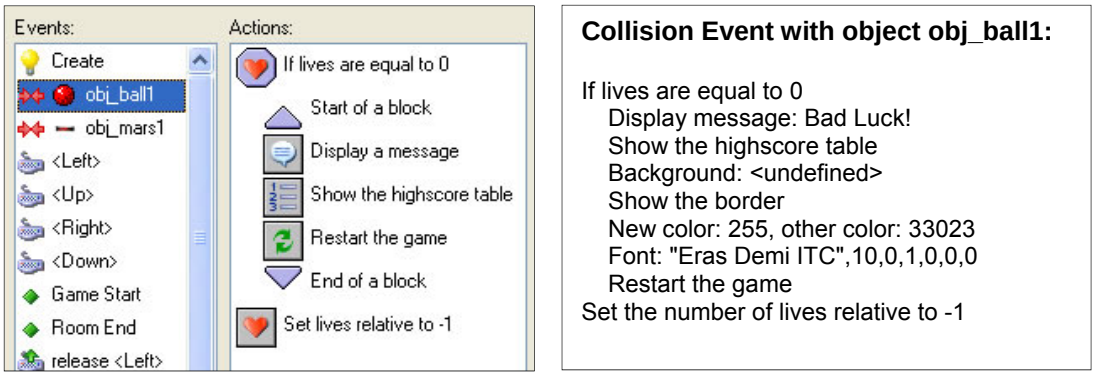


Figure 23: Game Maker's visual and textual information (ACJC)

Pupils learned about the importance of sequence when, for example, errors in the sequence in which events were ordered meant that the game did not function as intended (JDMB) and when the sequence in which rooms were ordered in the resources tree affected which room was displayed first when the game was run (ACJC).

7.2.2 Events

In *Game Maker*, all program interaction is achieved by selecting events (user inputs such as a key press, or non-user inputs, such as a collision between two objects). When an input occurs, an output follows. In learning to use these events, pupils were introduced to the idea of event-driven programming and to the key patterns of interaction in a game program (see Figure 24). Pupils quickly became used to selecting and referring to events.

Figure 24: The event selector

Pupils found it easier to understand those events which are user-activated (i.e. *keyboard/mouse* events), than those which are not (i.e. *step* and *alarm* events). Figure 26 shows the number of games which featured each type of event.

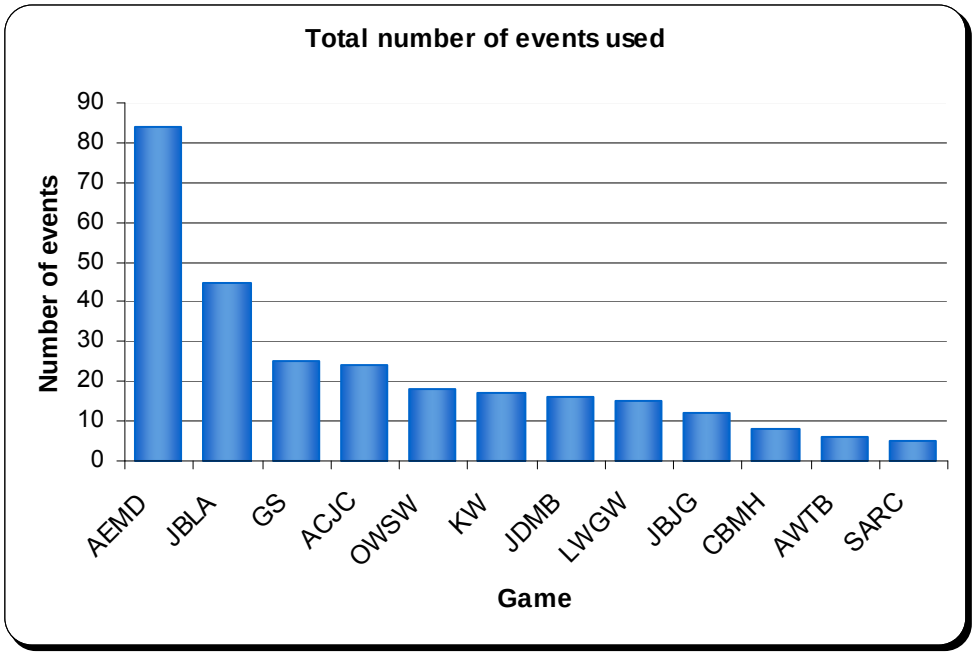


Figure 25: Number of events used in each game

The average number of *different* event types used in each game was 5, although, as Figure 25 shows, the total number of events used in a game was greater (average total number of events used = 23). The most frequently used event was the *create* event (see Figure 26), commonly used to set an object in motion when the game is run, or to set variables (such as score or lives) for it.

The *collision* event is the next most frequently used event - appearing in 10/12 games. This is not surprising, since much of the game play in the games authored is achieved by objects 'colliding' with each other on screen, and collisions are a core functionality of many adventure/arcade games, such as those created in this study. In playing games, pupils will have been used to the idea that when one object 'collides' with another, something will happen. In their own games, *collision* events were used as a mechanism to achieve a range of effects: to make objects disappear, to collect items and gain points, to decrease lives or score. Pupils learned that non-user events (such as collisions or alarms) function as game inputs, as well as user inputs, such as a mouse click or a key press. This expanded understanding of inputs was important learning.

Since the mouse and keyboard are common forms of input device and the arrow keys and space bar are commonly used when playing computer games, the use of these events was straightforward for most pupils.

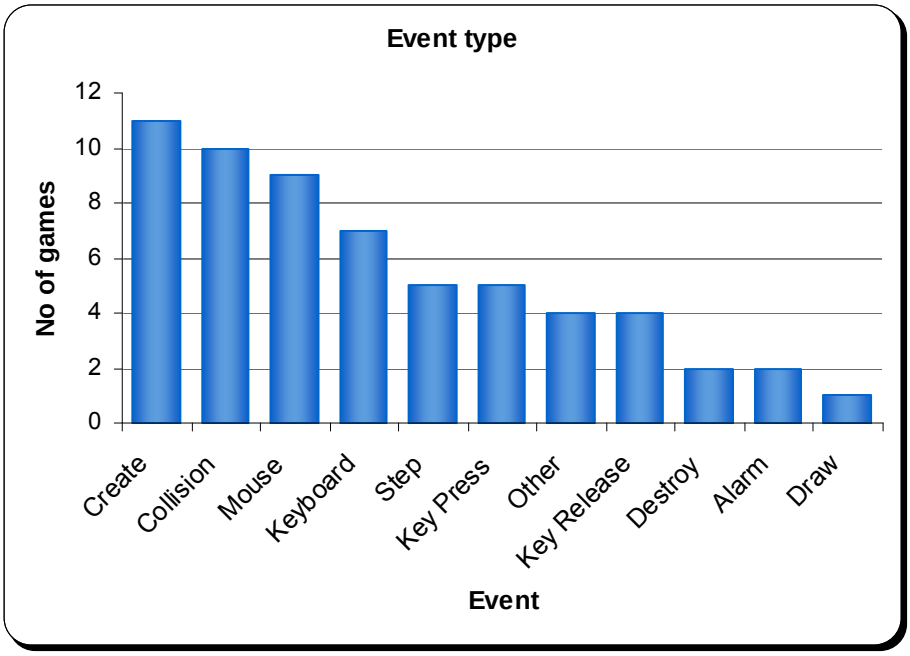


Figure 26: Type of events used in the games

The *mouse* event was used in 9/12 games, usually to click a menu button, or to navigate between screens. GS used the *mouse* event 16 times to enable the user to select 'start' and 'end game' buttons, and to access the 'help' and 'title' screens. He also used the *mouse* event to create a rollover effect for his buttons. All *mouse* events were used correctly.

Keyboard events were used in 7/12 of the games, typically to control the movement of the player character using the arrow keys. However, some pupils who used *keyboard* events (JDMB, SARC) had less success in controlling the stop/start movement of their player characters, since they did not implement an event to control the *stopping* of movement.

Key press and *key release* events were used in 5/12 games to control the movement of an object; *key release* events were used in 4/12 games to control the stopping of movement. KW successfully used the *key press/key release* events to control the start/stop movement of the player character using the arrow keys. The *key press* event was also used to create an instance of an object when the space bar was pressed, to give the appearance that the player character had thrown a stone or a missile had been fired (JDMB, AEMD).

The correct use of these events suggests that pupils understood the idea of simple, event-driven programming involving the concrete use of the mouse or keyboard as

inputs. Other events and non-user inputs such as the *alarm* event and the *step* event, are more abstract and these were used less frequently.

The *alarm* event is used to make things happen from time to time, without user input - for example, an object could periodically change its direction of motion. This event was used in two out of twelve games. AEMD learned to use the *alarm* event by following a tutorial and applied it to their game effectively five times. For example, they used it to set an interval between bullets firing:

> AE: We're going to add in an alarm so the player can't just hold down the space bar and shoot loads and loads of bullets, so we will set the alarm to 15 and once the alarm has counted down it will let you shoot another bullet.

The *step* event is used to execute actions continuously and occurs once every step (frame) of the game (30 times per second). This event was used in 5/12 games, most often to check values relating to object position. For example, OWSW used the *step* event to make objects on a scrolling background reappear at random positions on the screen when they had disappeared from view:

```
Obj_rocks
Step event:
If y is larger than room_height
   Jump to position (random(room_width), -120)
```

The *step* event was also used to repeat an action (JDMB, AEMD), such as destroying instances of objects once they have disappeared from view or creating objects intermittently, for example (AEMD):

```
Obj_enemy
Step event:
If y is larger than room_height+32
   Destroy the instance
   With a chance of 1 out of 180 perform the next action
Create instance of object obj_s_enemybullet at relative position (0, 16).
```

AWTB used the *step* event to continually check whether the player character in their platform game is in the air, in which case, gravity should pull it down:

```
Obj_Character
Step event:
If relative position (0,1) is collision free for only solid objects
   Set the gravity to 0.5 in direction 270
```

Else
  Set the gravity to 0 in direction 270

JBLA used the *step* event to execute a script to govern the movement of the player character:

Obj_player character
Step event:
Execute script script_shrane_mod4 with arguments (0,0,0,0,0).

Such use of the *alarm* and *step* events introduced pupils to the programming concept of repetition, and illustrated alternative mechanisms for controlling this pattern. Pupils learned that within the game loop, certain events occur continuously or repeat if certain conditions are met or game states are reached.

The *draw* event was used in one game (AEMD) to display the score, health bar and lives graphics on the screen, using coordinates to define their location:

Obj_controller_life
Draw event:
At position (0,404) draw image 1 of sprite spr_s_bottom
At position (180,440) draw the value of score with caption
Draw the health bar with size (12,449,138,459) with back color none and bar color green to red
Draw the lives at (16,420) with sprite spr_life

The *other* event incorporates thirteen events and was used in 4/12 games (KW, AEMD, ACJC, JBLA). Use of these events introduces the idea that game inputs are not only achieved by user input but also by game states (i.e. when there are no more lives, when a level is completed, when an animation ends).

AEMD used the *other* event correctly five times: the *outside room* event to destroy instances of bullets once they have disappeared from view; the *no more lives* event to launch the high score table once all lives are lost; the *no more health* event to reset the health value and to make the player character disappear once health is depleted; the *animation end* event to make an animation disappear after it has run its course, and to pause the game before the screen is redrawn and a new player character reappears:

Obj_explosion
Other event: Animation End:
Destroy the instance
Sleep 1000 milliseconds; redrawing the screen; true
Create instance of object playerplane at relative position (0,0)

> Set the number of lives relative to -1

ACJC used the *other* event twice: the *game start* event to start the game if the left mouse button is clicked, however, this event was used incorrectly - the *game start* event does not start the game, but defines what other actions will happen when the game starts; the *room end* event to add 20 to the score achieved at the end of a level. JBLA used the *other* event correctly twice - the *game start* event to set the player character's health value at the start of the game; the *animation end* event to make an animation disappear after it had run its course.

In so far as all pupils used events in their games, they learned about the concept of event-driven programming. They also learned that outputs can be controlled by user input, or by non-user inputs and game states. Whilst many events were correctly used, pupils also encountered problems with using events and these are discussed in Chapter 8.

7.2.3 Objects

In addition to learning about event-driven programming, using *Game Maker* introduces pupils to the concept of object-oriented programming (Overmars, 2004; Chamillard, 2006), a paradigm which sees program elements organised as objects, each of which holds its own behaviours and properties.

Pupils learned that in *Game Maker*, objects, rather than sprites (the visual appearance assigned to objects) hold programmed behaviour. This was a concept that pupils did not at first find intuitive - but which they grasped as they became more accustomed to using the software and the process of program generation it affords. Pupils did not initially understand that a sprite is simply an image, or why there had to be a sprite *and* an object.

> MD: There are some things that aren't really sort of logical in the first place, but you can understand them after a while … like having a sprite and then an object. I dunno, the sprites don't seem to do much on their own.

This idea that the visual appearance of a computer game is separate from the underlying program behaviour was new learning for pupils - but is a key concept in understanding how most computer-based systems are put together. This is important learning because children need to know about and experience the underlying

'constructedness' of digital media (Schelhowe, 2007). As *users* of technology pupils do not need to consider how systems are constructed or how they work. As *creators* of digital media, they learn what goes on 'behind the scenes' and this enhances their understanding of the technologies that surround them:

> SA: Yeah, 'cos when you play a game you just take it for granted, really, as something that just ... works. I didn't even know you could make a game. I've never had any experience of that ever.

In particular they learned that for the user to be able to interact with objects, they had to be created as separate entities. This was not immediately obvious to some:

> TB: I didn't realise you had to have rooms for the game to be made and have all the sprites and objects and have them all separately. Lots of different parts of it, that you have to build up layers to the game.

> LW: I should have used a blank canvas as my game background and then made black squares [for the maze walls] as a solid object and then placed them on my background so that the cat and mouse could not go in this place or off the screen. Instead I [drew the maze] in the background, which meant there were no barriers on where the cat and mouse could not go.

In the second extract, the pupil learned that to create a maze, the maze 'walls' have to be created as separate objects, and placed in the room 'on top' of the background graphic if they are to function as a barrier - the game background is no more than a graphic loaded into a game room to give it an appropriate visual appearance.

Those pupils who followed the print tutorials available learned about the concept of a 'controller object', which further enhanced their understanding of the nature of objects and their role in game design. The controller object has no visible appearance, and plays a 'global' role in the game. In contrast to other game objects, it has no role in the narrative of the game, but is used to manage game settings. For example, a controller object might be used to set variables, such as score and lives at the start of a game. Three of the games included a controller object. AEMD used this feature to set the score, lives and health at the start of the game, to show the high score table when all player character lives are lost and to display the life, health and score at the bottom of the screen. JBLA and ACJC used their player character to perform some of the

functions of a controller object - to play sound, to specify the appearance and location of the player character, to set the health, number of lives and score at the start of the game. In using a controller object, pupils learned that some elements (e.g. background sound, score, lives, room speed) are separate from the narrative interaction of the main game objects, and can be controlled separately.

7.2.4 Actions

Specifying the actions which objects should perform is the central programming task of creating a game in *Game Maker.* In using actions, pupils learned to construct their game program in individual steps and began to understand the use of functions in a computer program. They also learned about the common actions in computer games, for example, *'move in a direction at a specified speed', 'if the score is equal to 100, go to the next room/level', 'increase the score by 1', 'make an object disappear or reappear elsewhere on the screen'*.

Some actions were easier to understand than others. For example, the *move* and *go to next room* actions are straightforward and were used frequently without error. More abstract actions such as *test expression* and *set alarm* are more difficult and were used less frequently. However, the actions used in pupils' games do not necessarily reflect those that are easy to understand, for example, 8/12 games did not include the action '*play sound*', even though this is not a difficult action to understand or use. Neither does the use of a particular action necessarily mean it is understood or used appropriately.

Of the 92 actions available, the average number of different actions used in each game was 11. Fifty-three of the 92 actions were used across all games. The most successful game contained 169 actions and used 34 of the 92 different actions available.

The most commonly used actions were those which define object movement (*move fixed* (11 games), *jump to a position* (6 games), *jump to a random position* (5 games). Other commonly used actions were related to object destruction (7 games) or movement between levels (9 games). *Test* or *set score* and *lives* actions were used in 8 and 5 games respectively.

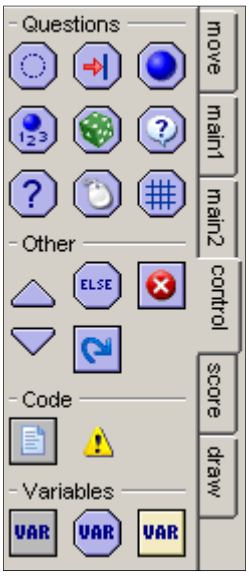


Figure 27: The control actions

Much of the learning conversation captured in the digital voice data is devoted to discussing which actions to apply - Figure 27 illustrates the graphical icons used to represent the control actions - and in solving problems arising from action selection, as illustrated in the following examples:

> SW: OK, first object is the snowboarder, the event is a collision with the rock and the action is that the crash sound is played and you lose a life and an animation of the snowboarder rolling off the snowboard.

> JB: Collision with the horse. Main 2, destroy. Self. Negative. All we need to do ... It needs to add points. How do we do this? Control? Score. We need to ... add life … draw life … test life … set life?

> MD: OK A., when you press the space bar we need somehow to shoot the bullet, so I suppose you create an instance of the bullet object.

Most of the actions used are pre-programmed. However, the *execute script* and *execute code* actions can be used to introduce pupils to writing functions themselves using *Game Maker's* textual programming language, GML. One pupil (JBr) used the *execute script* and *execute code* actions, sourcing a script from the *Game Maker Community* forum (Overmars, 2003). While he may not have understood all the code in the six page script he reused, he will have been able to understand some of it by reading the comments in green which accompanied the code (see Figure 28).

```
// Determine the motion speed based on the action:
//   N = walk_rate or run_rate px/sec
//   R = room_speed steps/sec
//
//   <spd> pixels   <N> pixels    1 second
//   ------------ = ---------- * ---------
//       step         second      <R> steps
//
//       spd      =     N     /     R

if action == "walk" {
spd = 100 / room_speed; // 100 pixels per second
}
else if action == "run" {
spd = 150 / room_speed; // 150 pixels per second
                                  }
```

Figure 28: Example of code comments

In using scripts created by third parties pupils began to see that code can be written in separate 'chunks' and this supported their understanding of modularity and code reuse. Using others' scripts also enhanced pupils' understanding of how particular effects can be achieved, such as in the example above, which governs the speed of movement of the player character.

Reusing others' code is accepted practice in the field of end-user programming (Kurland et al., 1987), but this approach was only used by one pair in this study. This was partly because the research school's internet policy restricted access to game-related sites and forums, but also because the *Game Maker Community* forum is not aimed at an educational audience and is neither accessible nor appealing to Key Stage 3 pupils. Another factor is that the pupils in this study were not accustomed to making independent use of online sources of support to find information at the point of need, and needed to be encouraged to seek out 'just in time' learning resources, as reported in Chapter 5. But importantly, such resources need to be designed for young people.

Moreover, there was little support for using the *execute script* and *execute code* actions in the commercial tutorials and resources provided. The *execute code* action is referred to only once in one of the textbooks made available in this study (Waller, 2009) and the other resources used (Giles et al., 2008; Jones and Wilson, 2008; Reeves, 2008) do not draw out the program's potential for teaching basic programming concepts, or introduce pupils to writing scripts in textual code.

However, with the advent of the new computing curriculum in September 2014, more structured use of the *execute code* and *execute script* actions would be a useful addition to Key Stage 3 units of work using *Game Maker*.

Parameters and arguments

In *Game Maker*, once an action is selected, parameters or arguments need to be set for it. Some pupils found this the most challenging aspect of creating a game:

> AW: When you drag [an action] across it comes up with an option about all the different settings that you can add to it and that's what's hard, because you've got to work out which settings it needs.

The idea that behaviours, such as speed and direction, have to be defined for an object in order for it to move was also new. In *Game Maker* these behaviours are defined as properties of an object, and involve pupils making decisions and having to think logically about the effects of those decisions:

> AW: What I mean is, when you drag [an action] over you've got to actually properly say what you want it to do … you drag the [action] across that you want, but it's actually putting the text into that box to say 'Actually, I want it to do this', because otherwise it's just pointless.

Setting the parameters for actions introduced more abstract concepts, such as whether a value is relative or absolute, for example. The concept of relative value was most often encountered in this study when pupils wanted to program a score mechanic for their games. They learned to set the score relative to its current value, rather than to an absolute value, and this was new thinking for some:

> GW: Do we want it relative?
> LW: Don't know what that means!

Yet it was not difficult for them to grasp, and they used the term appropriately in their working conversations:

> MD: Set the health bar relative to minus 5 for enemy colliding with enemy bullet.
>
> OW: We could do relative 5, relative 10? Like, it goes up by 10 each time, relative.

In nine of the twelve games relative values were applied to a variable, to add or subtract from the score (6 games), to subtract lives (4 games) or to decrease health (2 games). Relative value was also used to specify object position in five games (AWTB, JBJG, JDMB, OWSW, AEMD) and in setting the speed in one game (LWGW), where it was used in error.

Pupils learned that arguments need to be defined in many actions, for example to specify the direction or speed of a *move* action, or the position of an object. Much of their working conversation was concerned with what values to use in the passing of parameters and arguments and some pupils found this aspect challenging:

> TB: All the controls are quite complicated, the amount of different things that you have to put in … programming the sprites and the objects, it can be quite complex. And you have to know what it's talking about otherwise you can get it wrong and it may not work.

> JB: I used to [wonder why] computer games used to take so long to come out, and now I know it's 'cos … every little bit in there needs to have, like, loads of complicated things just to do that.

Pupils learned that arguments can also include expressions, which may also use relational and mathematical operators. Relational operators (<, >, =) were used in 6/12 games, often to test the value of a score, lives or x/y coordinates. Mathematical operators (+, -, /, *) were used in one game (AEMD) to test the coordinates of an object and to set the speed of an object.

Using actions taught pupils how many factors have to be considered when creating a game program and the importance of precise, logical thinking in setting arguments and parameters. It also developed their understanding of the structural patterns used in programs, such as conditional statements, loops, and variables. However, to support the learning of these concepts the terms themselves need to be emphasised and their use modelled in teacher-led interventions. Pupils need also to be encouraged to view the textual information for actions, since this gives them some exposure to how parameters and arguments are used in textual programming languages.

Selecting actions and setting arguments and parameters for them was a new practice for pupils and the problems they encountered here are discussed in Chapter 8.

7.2.5 Conditional statements

The conditional statement (If/If…else) is a key programming concept which defines the selection of actions in a program if a particular condition is met (Fincher, 2006). Conditional statements are achieved in *Game Maker* by selecting one of the *test* or *check* actions which test or check a game state and then trigger one or more actions if the condition is evaluated as true (see Figure 29).

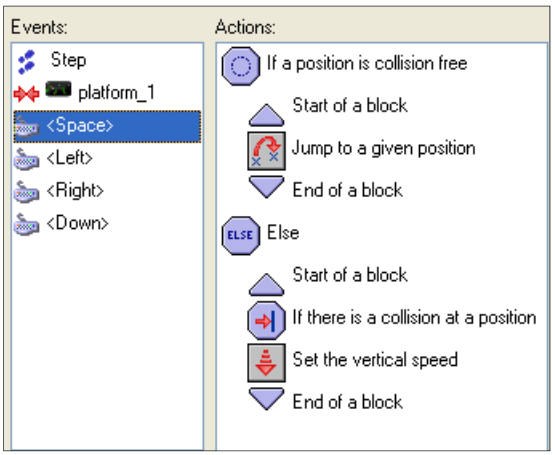


Figure 29: A conditional statement (AWTB)

Pupils learned that computer games are full of conditional logic which defines how the game play unfolds *('if the left arrow key is pressed, move the character left', 'if the player character collides with an object, increase the score by 10'*). All pupils in the study used conditional statements in planning their game interactions using an 'If … then' construct, as shown in Table 5. Observation notes and planning documentation suggest that this concept was straightforward for all the pupils in this study since all pupils completed these tasks with some success.

Computer games are a good vehicle for teaching the use of conditional statements in a motivating context (Adams and Webster, 2012) and *Game Maker* gives good support to the learning of this concept (Chamillard, 2006). Fifty-four conditional statements in total were used in 6/12 games. Eleven of the possible sixteen conditional statements were used across all the games. The most common conditional statement used in the games was the *test variable* action, which was used in 4/12 games; t*est lives* and *test score* actions were also used in 3 and 2 games respectively.

If	Event (input)	Then	Action (output)
If	Game starts	then	Enemies start moving down screen Background music starts Clouds 1,2,3 appear Player's plane appears
If	Player collides with enemy plane	then	Destroy instance of enemy plane Set health bar relative to -10 Play sound 'explosion' Create instance of object 'explosion 1'
If	Player collides with enemy bullet	then	Set health bar relative to -5 Play sound 'explosion small' Create instance of object 'explosion 2'

Table 5: Planning the game interactions using conditional statements

The *test variable* action was most often used to check the position of an object on the screen (OWSW, AEMD, AWTB) to see if it had passed beyond the boundary of the room, in which case, it would reappear at another location, to remain visible, as in this example (OWSW):

Obj_tree
Step event:
If y > room_height
  Move to position (random(room_width), -65).

This action was also used to constrain the movement of an object (JBJG), as in the following example where a horse can move up and down within a grassed area, but not beyond it:

Obj_horse
Keyboard event for <Up> key:
If y > 224
  Move relative to position (0, -4)

Keyboard event for <Down> key:
If y < 390
  Move relative to position (0, 4)

While 6/12 games included at least one conditional statement, only one pupil refers to them in the transcript. In this example he describes the use of a conditional statement to test the value of a variable:

> MD: A variable constantly asks a question and when it is true, allows an action to be performed. For the left key press event we [test] a variable – 'If *x* is larger than 40'. Then we can tell the car to move relative to -4 on the *x* axis (this moves the car left).

However, some pupils found it difficult to implement a conditional statement, (see Chapter 8), which suggests that the construct needs to be modelled and the term introduced explicitly, so that all pupils can understand and apply this structure in their games.

7.2.6 Loops

Another concept which pupils encountered is that some processes within a program need to be repeated. This is achieved using a 'loop' construct when there is a requirement for an action to repeat, or for a state to be continually checked in a program (Fincher, 2006). Pupils used the following mechanisms to create a loop-like structure in their games.

Game Maker operates a continuous loop during game execution and by using the *step* event, pupils can specify what actions they want to occur in each step of the loop (Chamillard, 2006). Six pairs in this study used the *step event* for this purpose. For example, OWSW used the *step* event to make an object reappear after it had disappeared from view. In this case, the *step* event checks the position of the object every second and relocates it every time it disappears beyond the visible area of the screen:

> **Obj_snowboarder**
> **Step event:**
> If y is larger than room_height
> Move to position (random(room_width), -65)

Another method for repeating an action is to use a conditional statement which allows code to be executed repeatedly based on a Boolean condition (true or false) (Kuruvada et al., 2010a). Six pairs used this method to achieve a loop construct.

In this example (AEMD), if the player character object (an aeroplane) exists, an enemy plane should fire bullets at it, throughout the game, but if no plane exists, the bullets should fire straight ahead:

Obj_enemy bullet
Create event:
If number of objects obj_playerplane is larger than 0
  Start moving in the direction of position
(obj_playerplane.x, obj_playerplane.y) with speed 8
Else
  Set the vertical speed to 8

Using these mechanisms, pupils learned about the concept of repetition and how it can be useful in a game program.

7.2.7 Variables

A variable is a named reference for storing a value in a program (Fincher, 2006). The use of variables is important in computer games, since the player's score, a character's health, an object's speed, direction and position have to be defined and stored in the game for it to give meaningful game play, and thus game making is a good context in which to teach the use of variables.

In *Game Maker*, several commonly used local variables (x, y, speed, direction, gravity) and global variables (score, lives, health) are inbuilt - they do not have to be declared as is normally the case in textual programming languages. Although this makes their use straightforward, it 'hides' the underlying concept. All pupils used at least one in-built variable, however, they may not have been aware that this was what they were doing because they only encountered the term 'variable' if they wished to test a value, as in the following examples:

> MD: For the left key press event we [test] a variable 'If x is larger than 40'. Then we can tell the car to move relative to -4 on the x axis (this moves the car left).

> OWSW: To make our new levels we are going [to test] a variable stating 'if the score is greater than 100 then switch to [level] two'.

It was not difficult for pupils to understand that certain values such as score and lives need to be stored in games, but they needed to be encouraged to refer to these as variables. The idea that variables can be created is a more abstract concept and only one pair (AEMD) attempted to do this:

> MD: OK, so you create a variable from the control menu, then you have to click the 'test variable', 'can shoot 1' and then you have to create a block, and then

> [test] another variable, 'if score is larger than 400' … points … create an instance of ... [the bullet object].

These pupils used variables with increasing confidence, and were able to do so because they worked through a tutorial. They used nine different variables in their game (lives, health, score, vspeed, speed, x/y), tested variables (room height/width) and created one. Their growing understanding of the role of variables is illustrated in the extract below, where MD identifies a need for an easily accessible list of variable names, so that he knows how to refer to the variables he wishes to use:

> MD: I did have a bit of a problem with ... 'cos you know ... there are variables and there are, like, rules that you can use, like 'room_width' is like ... a variable that is constantly changing isn't it? It can change, but there's nowhere in *Game Maker* that you can find [the names of] all of these. A. … just randomly typed in 'lives' and it just happened to be one. It could have been 'life', it could have been 'player lives' or whatever, but it just happened to be correct and there's nowhere you can find them out and I tried going onto the help files and it did list a few but it didn't list them very well.

This pair also referred to the use of variables in their planning documents:

- If the player collides with a fuel can, set variable 'fuel' relative to +10.
- We will need to test whether the player has collected all of the keys. If variable 'collectkeys' is equal to 4, then go to next room.

Whilst all pairs used variables in their games (see Figure 30 and Table 6), only two pairs referred to them as variables in the data. Not all variables were set or tested correctly (see Chapter 8 for a discussion of the difficulties pupils had with this). This suggests that teacher intervention is required to introduce the concept of variables, draw out the way the software handles these and refer to the in-built variables commonly used in games, since it is unlikely that the concept will be used or understood correctly without instruction.

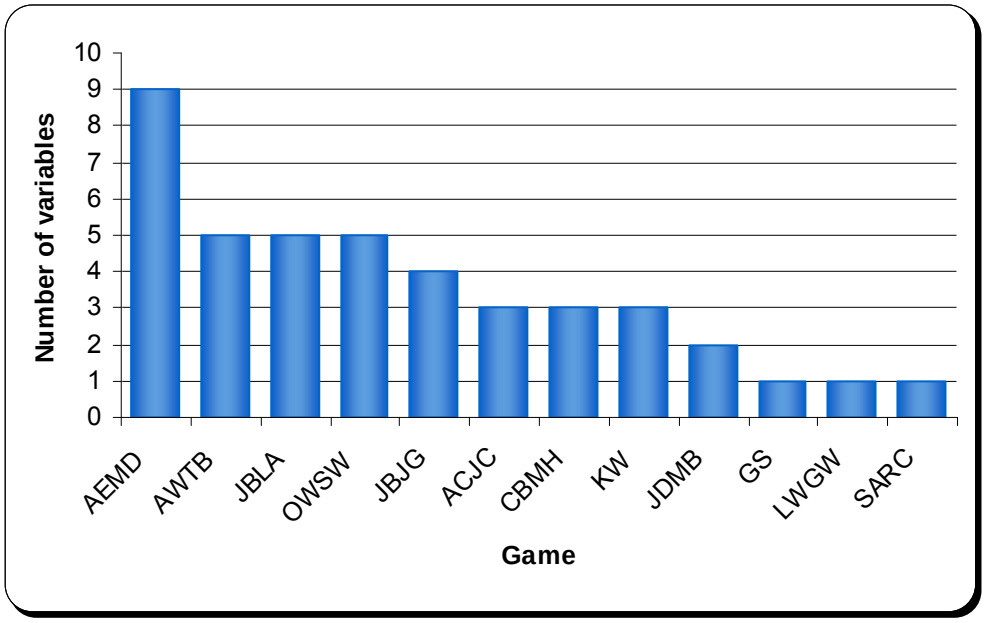


Figure 30: Number of different variables used in each game

Variable	Games used
Depth	2
Gravity	2
Health	2
Lives	5
Room height/width	2
Score	7
Speed	12
x/y	4
Create variable	1

Table 6: Type and frequency of variables used

7.2.8 Use of mathematical concepts

As well as learning about the programming concepts described above, pupils also learned that some mathematical concepts are important in game programs and that these are often used in setting the parameters, arguments and expressions of an action. This section describes the mathematical concepts they used.

Coordinates

The pupils in this study will have used coordinates in other subjects, but using coordinates to define the position of moving on-screen objects was a new application of that knowledge. Pupils learned to conceptualise the room, rather than the screen, as the game space. That game space is mapped by coordinates, measured in pixels (a standard room is 640 x 480) and knowing why that was important in terms of game programming was new learning for them. They learned that an object's position is defined by *x* and *y* coordinates (see Figure 31). They learned how to use these coordinates to prevent objects from disappearing from view and to make objects reappear, once they had travelled off the screen (if the *x* coordinate is greater than 640 or less than 0, or if the *y* coordinate is greater than 480 or less than 0, the object will not be visible within the game space).

They also learned that positions outside the room are valid locations; objects still exist and function outside the room. This learning is evidenced when pupils relocate objects to beyond the viewable screen area to make the reappearance of an object in a scrolling game more realistic, for example.

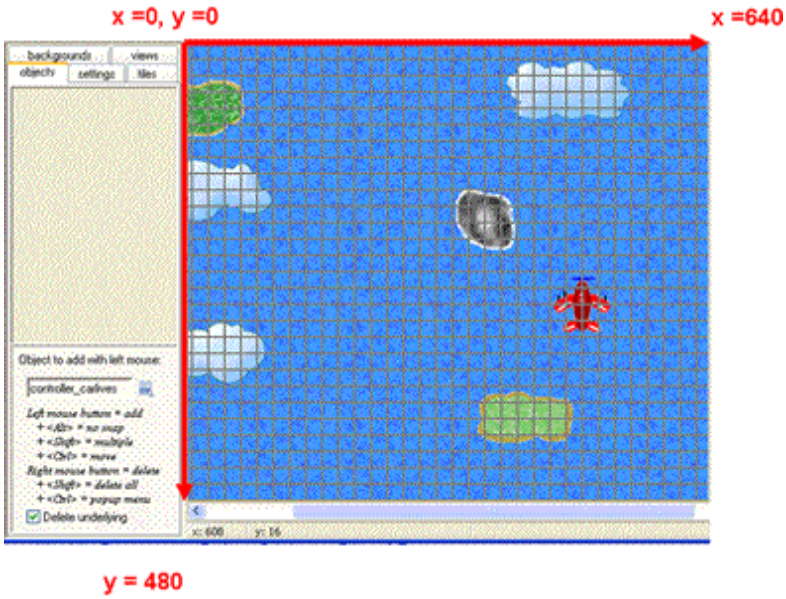


Figure 31: Coordinates in Game Maker

Several of *Game Maker's* actions involve the use of coordinates to specify the *x* and *y* position of objects in the game space. Coordinates were used in 9/12 games to define object position, to indicate the screen location of the health or lives graphics, to check if a location is empty, or to move an object to a particular position, as in the following examples:

To specify the position an object should move to (OWSW):

Obj_player character
Collision event with obj_tree:
Set the number of lives relative to -1
Move to position (320, 48)

To specify the location on screen of the lives and score graphics (AEMD):

Obj_controller
Draw event:
Draw the lives at (16,420) with sprite spr_life
At position (180, 440) draw the value of score with caption

Developing screen-based media where spatial boundaries have to be mapped and object position needs to be specified using coordinates was new learning for pupils.

Angles

Another mathematical concept that pupils met in a new context was the use of angles to specify direction of movement. In *Game Maker* angles range from 0-360 degrees as normal, but an angle of 0 refers to a direction to the right, 90 refers to a direction vertically upwards; 180 refers to a direction to the left and 270 points vertically downwards, as shown in Figure 32.

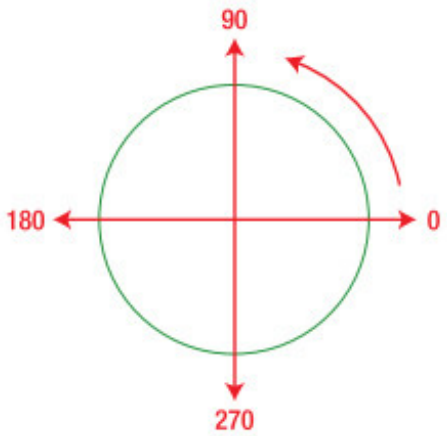


Figure 32: Angles in Game Maker (Habgood et al., 2010: 9)

For most of the time the use of angles to control direction is hidden behind *Game Maker's move in directions* action, where pupils set direction of movement by selecting directional arrows. But in other cases angles need to be specified. Angles were used to define direction in 4 games, such as in the following example (AWTB) where an angle of 270 is used to determine a downwards movement:

Obj_player character
Step event:
If relative position (0,1) is collision free for only solid objects
  Set the gravity to 0.5 in direction 270

Negative number

Negative number was used in 8/12 games for several purposes: in 3 games to refer to direction of movement, where a negative value equates to a move down (*-y*) or to the left (*-x*); in 3 games to define object position and in 5 games to decrease the value for score, health or lives variables. Negative number was also used to specify the depth (2 games) and the vertical speed (2 games) of an object:

Obj_snowboarder
Keyboard event for <Left> key:
If x is larger than 30
  Move relative to position (-4, 0).

Collision event with obj_tree:
Set the number of lives relative to -1

Obj_tree
Create event for obj_tree:
Set the vertical speed to -4

Step event:
If y is larger than room_height
  Move to position (random(room_width), -65)

Randomness

Pupils will have met the concept of randomness in mathematics, but in making a game they learned that randomness and probability can be usefully applied to enhance game play, by lessening predictability in a game. The idea that random behaviour can be programmed was novel to pupils.

Three pairs used random values to control the reappearance of objects on screen once they had disappeared from view (JBJG, OWSW, AEMD). They learned to set the *x* coordinate of the object so that it reappeared at random positions across the room width, to achieve a less predictable pattern of objects:

> MD: The *jump to given position* function is set to *x* = random (room_width) and *y* = -50 … [so] that the cars appear in random positions above the screen. This eliminates the look of repetition that games can sometimes have.

Randomness was used to define object position in 5 games, for example, when an object was set to *jump to a random position*, following a *collision* event. One pair (AEMD) used the *test chance* action to randomise the creation of an enemy bullet, so that this could not be predicted by the player.

Boolean logic

Pupils also learned that Boolean logic is used to define certain object properties, using true/false values. In 9/12 games true/false values were used to define an object as 'solid'. This property is commonly set as 'true' for non-moving objects and 'false' for moving objects. In four games true/false values were used to define whether a sound should loop or not. In one game a 'true' value was selected to redraw the screen after a pause in the game. 'True' values were also used in two games to define an object as persistent. In all the games 'true' values were used to define an object as visible. In two games (JBLA, AEMD) this property was set to 'false' to make an object invisible (e.g. a controller object). The idea that such properties have to be specified was a new way of thinking for pupils and strengthened their understanding of the precision and detail required in constructing computer programs.

Boolean logic is also implied in the use of conditionals where a condition is evaluated as either 'true' or 'false'. This binary construct is a common feature of multiple computing processes and becoming aware of its various applications developed pupils' ability to think computationally.

7.2.9 Program organisation

In creating a game, pupils were not only introduced to programming concepts, but also to programming practices relating to program organisation.

Naming conventions

In *Game Maker*, prefixes such as *spr_<name>, obj_<name>, back_<name>* are used to name and identify different types of game components. The resources used in the study introduced pupils to *Game Maker's* naming conventions and 7/12 pairs used these effectively some of the time. One pupil (KW) used the component type as a suffix instead of a prefix on some occasions. Eleven of the twelve games contain at least one unnamed sprite or object. One pair did not correctly name any of the sprites they used. Some pupils included hyphens in the resource names (spr_enemy-s; obj_enemy-s), which made them invalid. Pupils did not initially understand the need for correctly naming their game components or realise that this helps when managing game assets,

when referring to objects in the game program and when reading/checking program code.

Code commenting

Some pupils learned about the practice of code commenting, which involves adding comments to program sections, to clarify what the section does. One pair (AWTB) added a *comment* action to their game to remind themselves what the code meant:

Obj_player character
Step event:
COMMENT: Check whether in the air
If relative position (0,1) is collision free for Only solid objects
  Set the gravity to 0.5 in direction 270
Else
  Set the gravity to 0 in direction 270
COMMENT: Limit the vertical [sic] *speed*
If vspeed is larger than 12
  Set variable vspeed to 12

Another pair (JBLA) reused code which contained comments to clarify it:

```
// The direction the sprite faces (down, left, up, right)
direction_faced = "down";

// The current action (none, walk, run)
action = "none";
```

The use of code commenting was not included in the scheme of work, but pupils would have benefitted from learning about this programming practice, both in terms of reading others' comments to help them understand code, as in the example here, or in writing comments to document their understanding of the code they produced themselves. Adding code comments encourages pupils to read and check their code more closely and gives them useful practice in understanding and explaining their programs, an important part of learning to program (CAS, 2012a).

File formats

In constructing a game, pupils learned that there are two versions of a game file - an editable .gmk file and a non-editable .gmd file. Finished games can be saved as executable files which run independently of the *Game Maker* software. They learned to export graphic and sound files in suitable formats so that they could be imported into *Game Maker*, and that to save a file in a native, editable file format (e.g. .png, .mix) is different to exporting a file in its final, non-editable format (e.g. .gif, .wav). They learned that when a digital file is completed and no further editing is required it is 'exported' into

a non-proprietary, non-editable format, which is smaller in file size. This is important in the creation of computer games because file size affects the run-time efficiency of the game program, and this became a problem for one pair (AEMD) whose game featured 3 levels, multiple assets and took two minutes to load. Such learning enhanced their understanding of important digital literacy concepts in how digital media are created.

7.3 Computational thinking

In creating their games pupils learned about the basic programming concepts described above, but also encountered more general computational thinking concepts in context, and these are referred to throughout Chapters 6-9. Computational thinking emerges as an important element of the Key Stage 3 Computing curriculum (CAS, 2012a; DfE, 2013c) and is widely referred to as a 21st century skill of benefit to all (Wing, 2006; Wing, 2008; Perković et al., 2010; Repenning et al., 2010; Barr et al., 2011; Brennan et al., 2011; Denner and Werner, 2011; Google, 2011; Kane et al., 2012).

The term has received renewed attention since Wing's widely referenced article appeared in 2006 (Wing, 2006), and the computer science education community has since embraced its tenets and promoted its inclusion in emerging computer science curricula (Howland et al., 2009) and beyond (Perković et al., 2010), but it was Seymour Papert's work which first introduced the concept (Papert, 1970; Papert, 1980b) and later the term itself (Papert, 1996b). Papert's theory of constructionism developed alongside the introduction of computers in schools. For him, computers could alter and possibly improve the way people learn and think (Papert, 1980b: 208) and in this context 'computational thinking' refers to the type of thinking that is involved when working with computer systems, although a current definition of the term is the subject of some debate (Selby, 2013).

The context of computer game authoring has been identified as one way of introducing computational thinking to young learners (Kuruvada et al., 2010b; Repenning et al., 2010; Denner and Werner, 2011; Denner et al., 2012; Kane et al., 2012) and the following section illustrates that in creating their games, pupils are beginning to think computationally, as they use the language and practices of programming.

Pupils' understanding of programming languages was expanded as they learned that *Game Maker's* drag and drop action icons can be used to create programs and provide an alternative notation to those they had already met. Their understanding of the applications of programming was also expanded as they saw that programs are used to control virtual, as well as real-world systems:

> MD: We haven't done *Game Maker* before, we haven't done programming, well we've done *Flowol*, I s'pose that's a bit similar, but it's a bit more advanced than *Flowol* I suppose. I don't know, well in *Flowol* you do, you don't really make a game, you make systems that control ...

They also learned that there is an overlap with some of the programming concepts encountered in *Flowol* and *Game Maker* (input/output, sequence, loops, variables), and a difference in emphasis of some elements. Whereas the concept of loops and the use of the term itself is explicit in *Flowol* (where the idea of a feedback loop is important to the control of the systems presented in the software), in *Game Maker*, their use is more implicit, achieved in the selection of the *repeat* action, the *step* event, conditional execution, and the game loop as a whole. In *Game Maker*, the concepts of events, actions, user-interaction and collision detection are more strongly supported, because these are the programming constructs important for a computer game. For the pupils in this study, new ways of thinking were seeded via the understanding that different programming languages (and paradigms) share common ground, that different programming languages are suited to certain types of program, and that certain programming constructs are more important in some types of programs, than others.

Problem solving

In the context of ICT education, the practical tasks which pupils complete are commonly referred to as 'problems'. In this study the problem was to design a computer game for a particular audience and the activities pupils engaged in (e.g. devising game narratives, planning the game interactions, drawing storyboards, constructing the visual program) were all part of designing a solution to that overall problem. Within those separate activities pupils also had to solve actual problems as they arose. Chapters 6 and 8 give a detailed account of the generic and particular problems pupils in this study encountered in designing and programming their computer games. However, in relation to the focus of this chapter, the data indicate that the two key processes pupils needed to learn in terms of developing problem-solving skills were i) to decompose problems into smaller sub-problems and ii) to adopt

more systematic approaches to solving problems so that their responses were strategic and precise rather than generalised and haphazard. These findings are consistent with observations relating to the difficulties young people have with learning to program reported in the literature (see section 2.9).

Modelling

Modelling is the process of developing a representation of a real world system, or situation, which captures those aspects that are important for a particular purpose, while omitting everything else (CAS, 2012a).This aspect of computational thinking is evidenced in the data when pupils represented their games as a storyboard (see Chapter 6) and planned object interactions to 'model' their games (see Chapter 8). The findings in those chapters indicate that this process of abstraction was challenging for pupils because they did not 'see' the detail required, even at the level of modelling. Pupils were also engaged in modelling when they used *Game Maker's* visual programming environment to construct their games. The graphical action icons represent the events and actions they perform and these actions together constitute a visual model of the game and its underlying program code. The games they created are themselves 'models' of the real world.

Modularity

In programming their games, pupils were introduced to the idea that their game was a modular system, consisting of separate, interacting components (e.g. sprites, objects, rooms, backgrounds, sounds). They learned also that the game program as a whole was made up of, and could be broken down (decomposed) into separate 'sub-programs', each of which controlled a particular aspect of a particular entity. Just as the appearance of the game on screen was made up of separate layers, they learned that the graphics they created were composed of individual pixels. Learning to think of a computer game as a modular construct was new to pupils:

> TB: Well I didn't realise you had to have rooms for the game to be made and have all the sprites and objects and have them all separately. Lots of different parts of it, that you have to build up layers to the game … I didn't realise that you had to add an event and add actions to the event. I thought you would have just sort of one event and then just add lots and lots and lots of actions to that one event. I didn't realise you'd have lots of events with lots of actions.

Developing an awareness of the modular construction of digital media was important learning for pupils because they had not encountered this aspect of computation before.

Abstraction

Pupils in this study engaged in three levels of abstraction (Cutts et al., 2012) as they transitioned between their original ideas, written in English, a more technical phrasing of these ideas (pseudocode) in their plans, and the final transition into graphical code. Pupils engaged in abstraction when they represented game action as a set of events and actions represented by graphical symbols. They learned about the abstract set of actions and their concrete instantiations in terms of common game design concepts (score, sound, lives), interactions (collide, bounce, jump to random position) and programming constructs (repetition, selection, variables).

Testing/debugging

As they created their games, pupils were continually testing them to see if the events and actions they applied to objects produced the desired outcome. Some pupils checked their code and identified obvious errors. However, observation notes record that generally, pupils were not systematic when trying to correct errors (see Chapter 8). Only one pair (AEMD) ran their game in debug mode to isolate errors. In another example, KW did not read an error message beyond the first line and so was unable to identify the error and solve the problem.

Pupils needed to learn to read *Game Maker's* error messages, which identify the reason for the error, the object where the error occurred, the event where the error occurred and the number of the action which caused the error, as shown in the following example (KW):

> FATAL ERROR in
> action number 1
> of Mouse Event for Left Button
> for object instructions_obj:
>
> COMPILATION ERROR in code action
> Error in code at line 2:
> Move Patrick around using the arrow direction buttons on the keyboard at position 2:
> Assignment operator expected.

Constructionist learning theory asserts that pupils need to be given the freedom to get things wrong (Papert, 1999a), since on a practical level, programming is a continual process of debugging - but Papert also argued that pupils need to learn to approach

errors as a source of information, rather than as a problem and this is borne out by the data here. The pupils in this study needed to be taught more explicitly how to read and respond to error messages as part of 'learning to learn', an important principle of constructionist theory (Papert, 1999a).

Systems thinking

Systems thinking at its simplest, involves understanding that computer-based systems have some sort of input, involve some sort of data processing and produce some sort of output (CAS, 2012a). The pupils in this research had been introduced to these concepts in Units 8.5, and 9.1, of the Key Stage 3 National Strategy for ICT sample teaching material (DfES, 2003a, 2003b) when they learned how sensors provide input data in computer-controlled systems, such as a green house or a theme park ride. Authoring a game consolidated and extended this learning in that it gave a more concrete, tangible example of system inputs, in terms of the events (keyboard or mouse clicks or other in-game inputs) which control game objects, and the concomitant outputs (actions). Pupils understood the concepts of input and output in this new context and 1/3 pairs used the terms explicitly in their talk:

> MD: The output would be destroying the enemy plane then set the health bar to minus 10, play explosion sound and…oh yeah, show the explosion gif.

> LW: The output will be that the cheese will disappear.

Because this was the first time that pupils had authored a computer game, they did not have a well-developed mental model of how such a system is constructed:

> AC: We naturally wouldn't have thought of [events and actions]. If you asked us to create a game we would probably just say arrow keys move forward, if you get this add 10 points, not 'Where does the thing that you collide with go?', not 'If you release the key will it carry on moving?' or stuff like that.

However, as they made their games their understanding of a game as a constructed system grew, as illustrated by the following extract from a pupil journal:

> MD: To create a room you click on the room icon ... For our game we needed three rooms, one for the scrolling shooter level, one for the car level and one for the platform level. The dimensions for the rooms can be adjusted, but they need

> to be kept consistent throughout, otherwise the room may get wider or longer when you change level. It is important to know the dimension of the room so that you can set barriers and make sure the player doesn't travel off the screen.
>
> In the second level, the player takes control of a car. The aim is to avoid the other cars which you are overtaking. If you crash into a car however, you and the other are destroyed. This is achieved by adding a collision event with the enemy car to the player's car object, then adding destroy actions and setting them to destroy the player's car and the enemy car.
>
> Another event needs to be created for the movement of the car. Then you need an action. Normally you would just set a movement action which would make the car travel forward, but in this game there is no wall to stop the car, so we need to use variables. A variable constantly asks a question and when it is true, allows an action to be performed. For the left key press event we place a variable 'If *x* is larger than 40'. Then we can tell the car to move relative to -4 on the *x* axis (this moves the car left).

This extract reflects confident use of the language/discourse of *Game Maker* and the emergent use of specific computational thinking practices and programming constructs - all aspects of systems thinking.

This section has shown how the process of making a game with *Game Maker* introduces pupils to computational thinking (see also Dalal et al., 2009; Howland et al., 2009; Kuruvada et al., 2010b), however, while visual languages are more effective in supporting understanding of basic programming concepts than textual languages (Zagami, 2008; Koh et al., 2010; Stolee and Fristoe, 2011), they may also hinder the development of computational thinking either because they obscure the underlying computation taking place (Schelhowe, 2007; Howland et al., 2009), or because the connection between visual programming languages and 'real programming' is not clear to novices (Parsons and Haden, 2007), or because the language limits exposure to some programming concepts (Murnane and McDougall, 2006), as was the case in the current study.

7.4 Summary

This section has discussed the programming concepts which pupils encountered as they created a game in *Game Maker*. Table 7 below summarises the number of games which made use of these and indicates that computer game authoring can be used with some success to introduce certain basic programming concepts.

Programming concept	No. of games	Comment
Program interaction (input/output, event driven)	12	All games contained events as triggers for game action (range = 5-84; mode = 11-20).
Functions (actions)	12	All games contained functions (actions) (range = 5-170; mode = 11-30).
Sequence	12	All games involved sequencing actions.
Variables	12	All games included at least one variable (speed, score, lives, health, position x/y, gravity).
Boolean logic (true, false)	9	True/false values were used in nine games to loop sound or to set objects as solid.
Coordinates	9	Used to define object location in nine games.
Relative/absolute value	9	Used in nine games to add or subtract values from score, health or lives variables; to set speed and specify position.
Negative number	8	Used to refer to direction, position or to set the value for variables (e.g. score, lives, depth, speed) in eight games.
Conditional statements	6	Half of all games included at least one conditional statement.
Loops	6	Five games included a *step* event as a looping structure. In one game the *alarm* event was also used to repeat an action.
Relational operators (<, >, =)	6	Used in expressions in six games.
Randomness	5	Used to define object position or random creation of an object in five games.
Angles	4	Used to define direction of movement in four games.
Logical operators (AND, OR, NOT)	1	NOT appears in one game.
Mathematical operators (+, -, /, *)	1	Used in expressions in one game.

Table 7: Programming concepts evidenced in authored games

The data suggest that while making a game in *Game Maker* can introduce pupils to basic programming concepts and practices, certain programming concepts, such as conditionals, loops, and variables need to be explicitly introduced and modelled if they are to be learned effectively. When engaged in projects where pupils are programming almost without knowing it, it is important that teachers draw out the knowledge that pupils have acquired (Good, 2011). The learning resources made available to pupils gave step-by-step instruction and referred to programming concepts, such as variables and IF statements in passing (see Waller, 2009), but those concepts were not explained in detail, suggesting that there is a need for more emphasis to be put on drawing out the underlying programming concepts in such resources.

Furthermore, whilst *Game Maker's* drag and drop environment expresses some programming concepts effectively (e.g. conditions), with others this is not the case (e.g. arrays and lists, data types). If these concepts are to be introduced to pupils they would need to be taught the correct implementation in GML, *Game Maker's* textual programming language.

These findings give support to related research surrounding the use of other visual programming languages to teach basic programming concepts (e.g. Lavonen et al., 2003; Meerbaum-Salant et al., 2011; Denner et al., 2012). In these studies, concepts were only learned when students were explicitly taught the concepts while they created projects that used the concepts (Meerbaum-Salant et al., 2011: 168). Other studies found that while some concepts may be learned without instruction, others need a formal introduction if they are to be used effectively (Maloney et al., 2008; Schelhowe, 2010), since, in creating a computer game, pupils learn basic programming concepts without necessarily being aware that they are using those concepts (Kuruvada et al., 2010a; Good, 2011). In particular, computer game authoring does not deliver the more complex concepts well without additional teacher input (Denner et al., 2012).

As with the current research, some studies found great variation in the extent to which pupils used programming constructs when making computer games (Bruckman et al., 2000; Maloney et al., 2008; Denner et al., 2012), and note that some pupils used only modest amounts of programming concepts. Other studies conclude that the games produced only illustrated an understanding of the *targeted* computer science concepts (Chamillard, 2006; Carbonaro et al., 2010). This suggests that schemes of work need to specify what programming concepts pupils should use in their games, to ensure that a range of concepts (from easy to hard) are included.

The scheme of work used in this study was structured following constructionist principles (see Chapter 3) and this chapter has detailed the programming concepts and practices which pupils used in the games they made using this approach. The data suggests that while some concepts can be learned using a 'learning by doing' approach, others need more direct instruction if they are to be understood and applied by all pupils. This finding extends and updates previous research which found that constructionist practices which favour 'bottom-up programming', 'bricolage', and 'exploratory' learning (Turkle and Papert, 1990) were less effective than more instructionist forms of computer programming education (Ben-Ari, 2001; Meerbaum-Salant et al., 2010). While some studies support the idea that 'bricolage' is a valid way to learn programming concepts for some learners (McDougall and Boyle, 2004; Stiller, 2009), others suggest that exploratory learning does not lead all pupils to an understanding of the structure and operation of a programming language or lead them to develop skills such as problem decomposition, planning or systematic testing and debugging; it can also lead to inefficient or frustrating programming experiences (Kurland et al., 1987).

Findings from the current study suggest that the level of programming knowledge pupils acquired is, in Pea and Kurland's terms (Pea and Kurland, 1984), Level ii - code generator. At this level, pupils can write simple programs following examples, read and understand someone else's program and detect and correct some 'bugs'. But there is less evidence of program planning or understanding of how to make programs more efficient. Most children can learn to write programs at this level their research found.

Whilst their research implies that this level of programming knowledge is not sufficient, my own findings suggest that making a computer game introduces pupils to some key programming concepts and develops their ability to think computationally. In terms of the time scale available, perhaps that is enough. Educational goals for programming need to be realistic and achievable, given that most pupils at Key Stage 3 receive 36 hours per year to cover a wide range of topics in addition to programming, and bearing in mind the fact that many practicing ICT teachers need further training to feel confident in delivering this aspect of the new Computing programme of study (Nesta, 2014).

The next chapter extends the findings of this chapter by discussing the difficulties that pupils had with programming their games.

Chapter 8 Problems with programming

8.1 Introduction

This chapter describes the programming difficulties identified in pupils' voice recordings, planning documents, interviews and journal entries. An analysis of the programming errors made in the games authored is also presented. Together, these analyses address the research question 'What difficulties do pupils have with game programming?'

Learning to program is considered to be difficult for students of all ages (Jenkins, 2002; Robins et al., 2003; Dagdilelis et al., 2004; Parsons and Haden, 2007; Hernandez et al., 2010; Saeli et al., 2011; Brennan, 2013b) and many programming environments and languages have been developed in an attempt to make it more accessible to beginners (see Kelleher and Pausch, 2005; Murnane, 2010; Saeli et al., 2011). Difficulties may arise in understanding how a computer system works, how to write the program (syntax, notation), how to use data structures (loops, conditionals) and because non-motivating contexts are often used to teach programming concepts (Good et al., 2007) or inappropriate teaching resources and methods are used (Teague, 2014; Maguire et al., 2014). Particularly troublesome for novice programmers is that they have to learn to communicate with a computer in a precise and unambiguous way (CAS, 2012a), and may lack problem-solving skills (Govender et al., 2014).

Although *Game Maker* was designed to enable users to create computer games without the need to learn a textual programming language, users nevertheless have to learn to 'program' their games using its visual paradigm. Thus difficulties can arise in programming any component of a computer game e.g. sprite, object, event, action, room. The difficulties identified in this section fall into three broad categories: i) programming concepts that are difficult to grasp, because pupils do not understand how computing processes are produced generally, and of how computer games are constructed in particular (conceptual difficulties); ii) difficulties that pupils experience because they do not know how to use the software (operational difficulties); iii) a lack of precise, logical thinking and a reluctance to check/test their program statements (computational thinking difficulties).

To identify the difficulties pupils encountered, the object information for their games was scrutinised and errors in syntax were identified and categorised. Games were playtested and a detailed commentary of functionality was made and then analysed. References to difficulties in programming were also coded in the transcripts of pupil voice data and interviews, and in the planning and other documents they produced.

It is of interest that boys reported they had difficulties in programming twice as frequently as girls. This is probably due to the fact that some boys were more ambitious in the functionality they designed into their games, so encountered more problems in programming it. Also of interest is the finding that those of above average or high ability referred to difficulties in programming 4 times more frequently than those of average or below average ability. Those of lower ability referred to programming difficulties less frequently possibly because they did not attempt to go beyond the basic level of complexity of the tutorials they had followed.

Figure 33 shows the coverage of comments in the transcript referring to programming difficulties, coded by ability and gender.

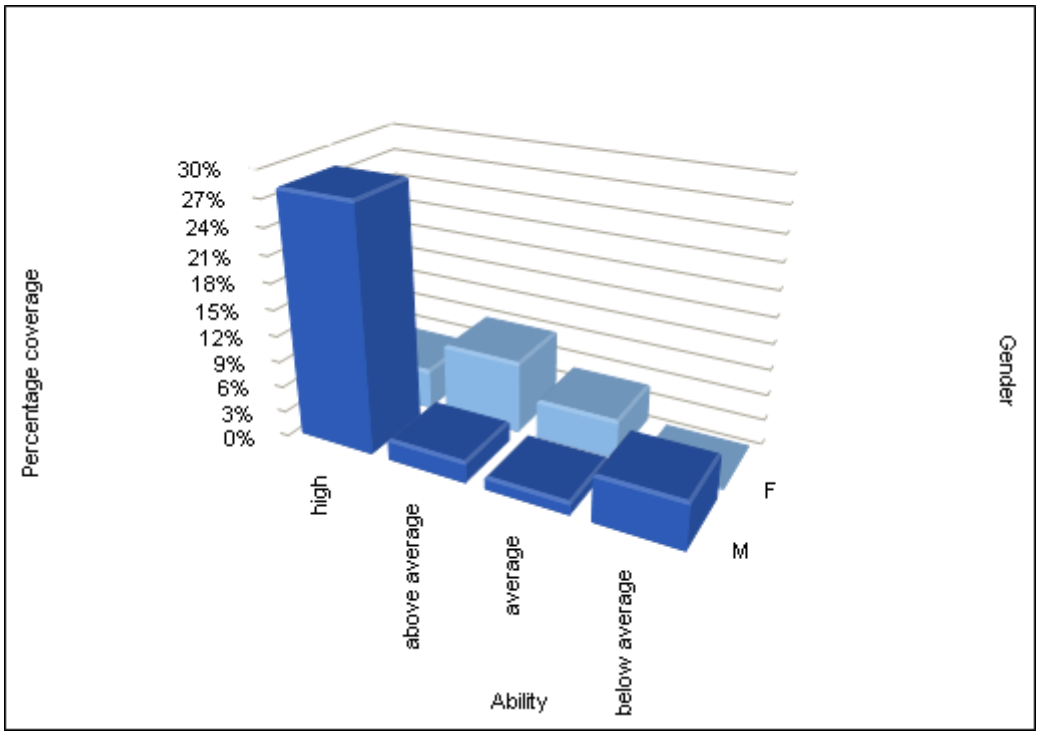


Figure 33: Programming difficulties coded by ability and gender

8.2 Program design

In general, there was a lack of program design which led to poor programming decisions. In particular, designing the algorithm for each object and planning the interactions between objects was not approached systematically, and was often superficial or incomplete. Although pupils had planned their games, they did not always use these plans later when they came to implement their game. In practice, much of their programming was achieved by 'bottom up' tinkering (Turkle and Papert, 1990) and for some, their enthusiasm to begin making their game meant that they were reluctant to spend time on planning tasks. This has implications for the pedagogy of game authoring. Whereas the 'learning by doing' approach promoted by the constructionist framework of this study may be effective for learning the software and making a basic game, a more structured approach is required for the program design stage. Pupils need to carefully consider the objects that make up the program, the actions the objects perform, the events which trigger those actions and the interaction between objects. In addition, more attention needs to be given to developing algorithms for the common processes which pupils require when programming games, before they implement the drag and drop code.

Analysis of the planning documents indicates that pupils found it difficult to conceptualise their games as a whole, and to decompose them into constituent parts. In particular they were not used to being precise and systematic in their thinking, because they had little prior experience of designing programs and were not used to thinking computationally.

This finding gives support to related research (Meerbaum-Salant et al., 2011), which observed that when faced with a programming task in *Scratch*, using an exploratory learning model, students did not approach it by thinking on the algorithmic level - instead they selected items that seemed to be appropriate for solving the task and then combined them into a script. Earlier research in programming in *Logo* and *Basic* (Kurland et al., 1987) also found that pupils using discovery learning approaches did not structure solutions, decompose problems or develop efficient algorithms.

Although pupils completed some of the planning tasks, this was not always done in sufficient detail or systematically. As a result they were less useful to pupils at the implementation stage. Some pupils forgot to use their plans:

> AE: Our game is quite complicated, and so it has quite a lot of actions and

> events. It took us quite a while to work out what the rules would be for our game. As we started to create the game, we discovered that we had forgotten some of the rules [in our plan]. However, this is not a problem, because it is easier to think of what the rules need to be as we go along.

This pupil's claim that it is easier to think of the rules for his game as he 'goes along', exemplifies pupils' preferred approach to developing their games, but while they preferred concrete, 'bricoleur' styles of working (Turkle and Papert, 1990), in practice they needed to plan events and actions for the objects in their games more systematically so that they had some sort of structure on which to build.

8.2.1 Language

In their planning documents, most pupils did not use the terminology they had encountered in the tutorials or the *Game Maker* program to help them define their interactions and this reduced the precision of their plans and made the planning document less useful to them. For example, KW, CBMH and JBJG do not use *Game Maker's* term '*collision event'* but use words such as 'collect', 'touch' and 'hit'. Because they did not use the terms available in the software, selecting the correct event was less obvious to them when they began to build their games.

The same was true when pupils referred to actions (see Table 8). Although the example given to pupils had modelled the use of imperatives to define actions (since this is how they appear in *Game Maker*) some pupils did not make use of this construct and as a result their plans were less supportive to them. JBJG refer to their player character 'running' instead of using the action names for movement ('move' or 'jump'); their horse 'disappears' when the correct term is 'destroy instance of horse'; the player character 'gains' points, when the action required is to 'set score relative to +10'.

Object	**Event (input)**	**Action (output)**
GS Ghost	Left click with mouse.	Ghost fades away when certain amount of points reached.
GS Ball of darkness	Left click held.	When left clicked and held, if Shadow hits ball -1 life.

If	Event (input)	Then	Action (output)
JDMB If	Left mouse is clicked	then	The object is thrown and if it hits a monster a scream sound is made.
JBJG If	When you get points to the end of the level	then	You move to the next one.
AWTB If	You press left	then	You go left.

Table 8: Use of language in pupils' planning documents

Others found using the language more straightforward. Four pairs (AEMD, ACJC, KW, OWSW) used the correct terminology to refer to their events and actions, as shown in Table 9, and this not only added precision to their plans, but will also have been more supportive to them when it came to constructing the game code.

If	Event (input)	Then	Action (output)
OWSW If	Player collides with blue snowflake	then	Score sound plays.
AEMD If	Player collides with enemy bullet	then	Set health bar relative to -5. Play sound 'Explosion small'. Create instance of object 'explosion 2'.
AEMD If	Right arrow key pressed	then	Player moves right at speed 3.

Table 9: Use of correct terminology in pupils' planning documents

Most of the 'errors' in their plans for the game interactions arise from a lack of systematic thinking. Some viewed the game plan as a description of 'what happens if…' and were not able to define the game in terms of its interactions - to specify user or game inputs (events) and the associated outputs (actions), even though this planning document was intended to be a scaffold for that.

These findings suggest that, although pupils may be reluctant to do so, more time needs to be spent on the design of the game interactions since this is the precursor to programming them. There is a need also to encourage pupils to use the language of *Game Maker* in the planning stages. It is likely that if more pupils had used the event

and action names available in the software at the planning stage, they would have later managed to 'code' their games more efficiently and effectively.

8.3 Programming concepts

Once they had completed their planning documents pupils began to implement their games and this brought them into contact with several basic programming concepts and practices. The following section presents the problems they encountered with these.

8.3.1 Sequence

One of the key concepts in learning to program is that commands need to be written in a logical sequence. Sometimes this caused errors for pupils. In the following example, ACJC set the actions for the control of the lives mechanic but did not sequence these logically:

Obj_Player character
Collision event with object obj_ball1:
If lives are equal to 0
  Display message: Bad Luck! Better Luck Next Time!
  Show the highscore table
  Restart the game
Set the number of lives relative to -1

The correct sequence would have been to use the *set lives* action to subtract a life from the current value before checking whether all lives have been lost. Similarly, JBLA made an error in the sequence of commands which managed the health mechanic:

Obj_Player character
Collision event with object Evil Dude:
Set the health relative to -7

Other event: Game Start:
Set the health to 100

In this case the health value does not decrease as intended because the *game start* event to set the health is listed after the *collision* event. Sequence was also a problem where the messages spoken by JBLA's 'Evil Dude' character appear on screen before the appearance of the character itself and when ACJC's title screen does not display because the rooms comprising the game levels are placed in the wrong order in the resources tree.

Such errors occur partly because the concept of sequence is less visually explicit in *Game Maker* than in other programs, such as *Scratch* or *Flowol*. Although there is an order in which the software will execute code, this is not apparent to the user - there is no visual representation of the run-time progress of the program, unless the game is run in debug mode. Therefore, pupils need to be encouraged to think carefully about sequence and to check it as a source of error in their programs.

8.3.2 Objects

In their planning document pupils were asked to detail the objects in their game and to list the events and actions for them. Eleven of the twelve pairs made some attempt to complete this section, following the example given in Table 10.

Object	**Event (input)**	**Action (output)**
Clown's head	Mouse event - left click.	Set score +10 points. Play sound 'Hit'. Head jumps to a random position.
Clown's head	Collision event with wall.	Head bounces in a random direction. Play sound 'Bounce'.

Table 10: Example initial planning document

This task introduced pupils to the need to decompose their game into its separate objects and interactions. At this stage most pupils' responses were characterised by a lack of completeness. In general, they did not complete the table for all objects, or all levels. Only 1/12 pairs included the events and actions needed for the button objects on their title screen. The events they refer to are mainly *collisions; create, keyboard* and *mouse* events are rarely featured. They do not list all actions for each object and specific details, such as the speed of movement, are not given.

In the games, a number of errors were made with objects. Some pupils did not have a secure understanding of the concept of 'objects' itself. For example, pupils were asked to create a title screen, which included one or more interaction buttons to start the game or to launch game instructions. Some pairs found this problematic because they did not understand that a 'start' button is an object which needs to be created and programmed in the same way as other game objects. Whilst they understood that game characters and other game items are objects, the idea that game navigation is

achieved by the use of objects (buttons) which require an input and an output was less intuitive.

Other problems arose because pupils did not have a clear understanding that in a game program all elements exist as separate entities. In the following example, a pupil has drawn the button graphics as part of the title screen background, instead of creating them as separate objects and so could not program them to respond to user input:

> TB: We've created the title page ... but we've just found out that we have to do the 'start' button and the 'introduction' button and the 'exit' button separately, so we've deleted the buttons and we're putting the background in separately to the 'start' button.

The same problem occurred for LWGW, who in creating a maze game, had drawn the maze as part of their background graphic, rather than constructing the maze out of separate wall objects.

Keeping objects in view

Within the game itself, pupils encountered several problems with objects. Keeping the player character and other objects in view on the screen was problematic for 9/12 pairs. Objects disappeared from view often because pupils had given the object a speed, which meant that the object continually moved at that speed, eventually disappearing from the screen. This was surprising for most pupils, who assumed that the edge of the screen was itself a boundary. They learned that object motion and position has to be controlled, either by using the correct *key press/key release* events to start and stop movement, by using or checking its coordinates, or by creating a solid boundary around the edge of the room.

For some pupils, controlling the movement of their player characters was difficult to achieve. In 3 games (LWGW, CBMH, JDMB) player characters move correctly when the arrow keys are pressed, but do not stop moving when the keys are released. It was not at first apparent to some pupils that just as setting an object in motion has to be programmed, so too does the stopping of movement:

> JC: We've got a problem at the moment that if we press an arrow key the person keeps on moving.

> JB: We had loads of trouble trying to get it to … stop at the end instead of going off screen.

In general terms, they did not understand that when writing programs, instructions have to be precise and unambiguous.

In four other games, the player characters are controlled effectively using the arrow keys, and in the course of play remain in view, but JBJG's horse disappears from the screen to the right/left if the right/left buttons are held down and JBLA's character disappears on the right, left and top of the screen. The movement of GS's, and ACJC's player characters is controlled effectively by the arrow keys, but they can travel off the screen in all directions, and return to view only if the opposite directional arrow key is pressed. In such cases, pupils needed to keep the character in view by comparing its vertical and horizontal coordinates with the room height/width so that movement stopped when these values were reached. This mechanism is not difficult to understand, but pupils needed instruction to implement it. OWSW and AEMD used this method to keep their player characters on screen and learned to do so by following a tutorial.

AWTB partially solved the problem of their player character travelling off the screen, by creating a 'wall' around 3 sides of the room, but their character disappears from the top of the screen when he jumps because there are no interactions set between it and the wall objects to prevent it from doing so.

Controlling object (dis)appearance

Since several games included a scrolling background, a commonly occurring problem was that objects disappeared from view and did not reappear as intended (see Figure 34). Five of the twelve pairs had problems with this.

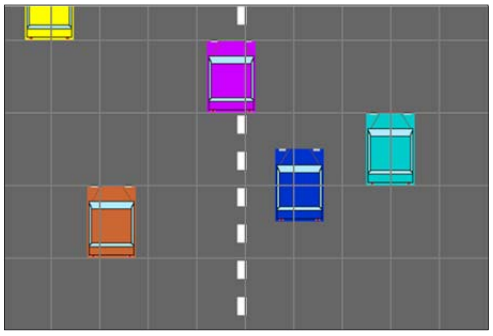


Figure 34: Objects disappearing from the screen

The collectable items (apples) in JBJG's horizontally scrolling game disappear once 'eaten' by their player character (a horse), but there is no mechanism to make other instances of apples reappear. After 5 seconds of game play no apples are visible, so the player cannot accumulate further points. Likewise, obstacles (logs) disappear from view after several seconds of gameplay and the horse is left running in an empty forest. In OWSW's game, objects disappear from the vertically scrolling screen after 7 seconds of game play, and there is nothing more for their player character to collect or avoid:

> OW: Our latest problem is that the objects don't reappear when they go out the screen so we will create an action and event that says 'if object is larger than room width, reappear at random position inside the room'.

Only 1/5 pairs (AEMD) managed to solve this problem, and to do so they used a conditional statement, in a *step event* (loop) as follows:

> **Step event:**
> If y is larger than room_height
> Jump to position (random(room_width), -120)

This code compares the object's *x* and *y* coordinates with the room width/height and relocates it within those dimensions once it has disappeared from view (i.e. when the *y* coordinate is greater than the height of the room the object will disappear from the bottom of the screen and then reappear from the top of the screen at a random position along the width of the room). Another pair (OWSW) attempted to use this solution, but did not implement it correctly.

The difficulties here arose because pupils were not used to thinking of the screen as a space mapped by coordinates, in which object position is defined by *x* and *y* values. They learned that to keep objects within this space they either needed to program objects to remain within the room's coordinates or provide some sort of boundary. However, such learning is unlikely to occur without instruction. This is important for the pedagogy of game authoring - there are certain core game functionalities which pupils need to know how to program, if they are to achieve a playable game, and these need to be introduced at the point of need.

Objects also disappeared from view due to other programming errors. For example, AEMD programmed *all* cars of the same colour to be destroyed when their player

character collided with a *single* instance of that colour car:

> **Obj_Player character**
> **Collision event with pink car:**
> Destroy the instance
> For all pink cars: destroy the instance

After several seconds of game play, there are no more 'enemy' cars on the screen, the player car travels along an empty street - and there is no way to end the level and progress to level 3.

Objects also disappeared from view because pupils did not yet understand the way in which *Game Maker* 'layers' objects on screen. JBLA's player character becomes 'hidden' underneath sections of a garden; JBJG's and JDMB's player character disappears behind other game objects. These errors occurred because pupils had not specified a value to define the 'depth' of these objects.

Sometimes, objects are intended to disappear, and not to reappear, such as when an object is 'collected', as in LWGW's game - where a mouse must 'eat' a finite number of cheese objects to gain points. LWGW correctly selected the *collision* event between the mouse and the cheese objects, but because they had created 7 separate cheese objects (instead of placing multiple *instances* of one cheese object in the room) they became muddled about which objects to reference in the *collision* events they created to make the cheese disappear. Thus multiple instances of cheese disappear at the same time, or cheese does not disappear at all.

Other pairs create objects and correctly set events and actions for them, but then do not place instances of the object in the game room, so they cannot function as intended (JBJG, SARC).

Controlling object movement

The most commonly occurring problems referred to in the data relate to controlling object movement.

Following the cursor

Some problems occurred because pupils simply did not know how to achieve certain effects and the tutorials and other resources did not cover these aspects. For example, two pairs (LWGW, JDMB) initially intended the movement of one of their game objects to be controlled by following the cursor:

> GW: We couldn't have our computerised cat chasing our mouse, we had to have it on random. Well originally we wanted it to be, like, following the [cursor] … but that was too complicated.

> JD: If we can't customise the brick to be thrown wherever the [cursor] is, maybe we can just have it thrown directly in front and it just keeps going as far as we want it until it hits the monster.

This mechanism is easily achieved by using a *step event* and adding either a *jump to position* or *move towards* action, and then attaching the *mouse_x* and *mouse_y* variables to the *x* and *y* coordinates of the object. However, pupils needed to be shown how to implement this behaviour; schemes of work need to include support for common functions in games, such as this.

Setting the object speed

Several errors occurred in setting the correct value for the speed of a moving object. Three pairs did not realise that in addition to applying the *move* action they had to specify a speed in order for an object to move. In CBMH's game, spiders are intended to move towards the player character, but no speed is set for that action, so no movement can occur. Similarly, LWGW do not set a value for the upward movement of their player character.

Others applied a value for speed when it was *not* required. ACJC used the *key release* event and *move fixed* action to stop the movement of their player character, but set a speed of 3 instead of 0. AEMD used the *keyboard* event to specify that the player character should *not* move when *no* key is being pressed, but set the speed to 8 instead of 0.

```
Obj_Player character
Keyboard event for <No Key> key:
If object is aligned with grid with cells of 24 by 24 pixels
   Start moving in directions 000010000 with speed set to 8
```

Pupils learned that to create the effect of scrolling movement a speed has to be set for the game room and that this has to be set in relation to the speed of the objects in the room:

> MD: I … noticed that when the player presses the down arrow, the player travels down the screen at the same speed as the scrolling background. This

gives the impression that the game has stopped. To rectify this, we would have to either increase the speed of the scrolling background, or decrease the speed of the plane.

Those who created maze games (KW, AEMD) learned that object speed has to be set to a value which is a factor of the grid size in order for correct motion to be achieved.

Jumping

Other pupils (AWTB, JBJG) who wanted to include a 'jump' action for their objects found this problematic, because they did not at first understand that they had to refer to the *x* and *y* coordinates of an object to manage vertical and horizontal movement:

> AW: We have had problems with making the character jump in the air and coming back down, so he is only in the air for a short amount of time instead of an unlimited time.

> JB: We couldn't get the horse to jump properly ... 'cos our horse on the background was going higher than the grass and into the sky so it looked a bit weird, but we managed to get it to do 'if x and y equal so and so, stop'.

They solved the problem by simplifying the horse's movement to approximate the action of jumping over a log; now that their horse could move up and down, they had to constrain this movement so that the horse would only be able to move within the grassed area of the game world:

Obj_Horse
Keyboard event for <Up> key:
If y is larger than 224
  Move relative to position (0,-4)

Keyboard event for <Down> key:
If y is smaller than 390
  Move relative to position (0,4)

Random movement

Three pairs (LWGW, CBMH, SARC) had problems with achieving random movement for their 'enemy' objects. Although the objects move in random directions they disappear from the screen after several seconds and there is no mechanism to return them to view.

In CBMH's game some instances of the spider object move in random directions; others remain stationary:

Obj_Spider
Create event:
Start moving in directions 111111111 with speed set to 1

This error arose because in selecting the directional arrows in the *move* action to achieve random movement, they had also selected the 'stop' option. LWGW made the same mistake with their animated cat object.

JDMB's monsters remain in their starting position because no events or actions have been assigned to set them in motion. In the example below, the first *move* action (*move to a random position*) is intended to reposition the object to a random location on the screen, but cannot function because it is included in a *destroy* event. The second movement action is intended to send the monster back to its starting position when it has been hit by a stone, but because the monsters don't move, this action can never execute:

Obj_Monster
Destroy event:
Move to a random position with hor snap 0 and vert snap 0

Collision event with object stone:
Destroy the instance
Move relative to position (0, 0)

These errors in controlling object movement largely arise because pupils have not yet learned that computers are deterministic and perform actions and processes only if they are explicitly and precisely told to do so. In other cases errors occur because pupils have not planned the interactions or checked the logic of their code.

8.3.3 Events

Planning events

Pupils were asked to plan the events and actions for their game objects in an initial design document. At this stage pupils were not systematic and did not detail all the events for all their game objects. The majority of pupils did not define what should happen when their player character first appears on the screen. Some pupils did not detail which events would trigger object movement. Most pupils only detailed the collisions in the game. Two pairs (LWGW, JDMB) struggled to understand what an event was at this stage, as is evident in the examples given in Table 11.

Object	Event (input)	Action (output)
LWGW Cheese	Stationary. Is able to be eaten and give points.	The cheese will disappear.
JDMB Man/woman	Stands on first level with bottle in hand.	Walks around castle wall.
JDMB Man/woman	Hiding behind pillars.	Peer outside pillar and through object.
JDMB Zombies	Diagonal movement.	When mouse clicks zombies collapse and are knocked out. Sound (scream). Score goes up every zombie.

Table 11: Misunderstanding events

Errors also occurred because there was some confusion between the meaning of the words 'event' and 'action', since in everyday usage, both can be used to refer to something happening - whereas in *Game Maker*, the word 'event' refers to a game input and 'action' refers to a game output. Figure 35 (an extract from AEMD's planning document) illustrates the confusion of terms.

Title Screen

Action	Event	Action	Event
Spacebar	Go to next room	F1 key	Show game instructions

Level 1 – Scrolling aeroplane shooter

Action	Event	Action	Event
Spacebar	Create instance of "bullet"	Arrow key	Move "plane" in a direction
	Set "bullet" vertical speed	Collision with enemy	Create instance of "explosion"
Collision with bullet	Set health relative to -5		Set health relative to -10

Level 2 – Scrolling car level

Action	Event	Action	Event
Left arrow key	Move "car" in a direction	Right arrow key	Move "car" in a direction
Collision with "o_car"	Set health relative to -10	Collision with "rock"	Set health relative to -5
Collision with fuel can	Set var. "fuel" relative to to +10	Collision with "oil"	Set health relative to -5

Figure 35: Confusing events and actions

A similar error is made when a pupil refers to an 'event' as an 'action' and vice versa in their working conversation:

> OW: When we make the start button graphic, which will be an object, we'll do an action that says 'If button ... if object is clicked', event 'jump to room 0' and the game will start, yeah?

These errors highlight the need to focus on keywords, such as ‘event’ and ‘action’, as an important element of learning to program in *Game Maker*.

Pupils sometimes describe events and actions for more than one object at a time, or generalise events, instead of listing each object and event separately (see Table 12).

Object	Event (input)	Action (output)
JBJG Apples and carrots	Disappear when touch horse.	Add more points; appear randomly.
JBJG Horse	Use arrow keys to move.	Move left, right, up, down.

Table 12: Generalising events and actions

Later in the design process pupils were asked to outline the interactions in the game in more detail, in a ‘Rules of the Game’ table. By this stage pupils had begun to separate the inputs and outputs for their player character and showed a better understanding of individual events and actions. At this stage also, they introduce non-user inputs for actions in the game (such as collisions, object locations, lives or score status), as shown in Table 13.

If	Event (input)	Then	Action (output)
KW If	Lose all lives	then	Message ‘Game Over’.
OWSW If	Score > 100	then	Next level commences. Screen scrolls faster.

Table 13: Use of non-user events

Nevertheless, analysis of these documents shows that pupils were still not able to conceptualise the game in the level of detail required and this is identified in the literature as one of the difficulties pupils have with programming (see section 2.9). Pupils do not define all events in their games, for example JBLA only include events for the movement of their player character; events for other game objects are omitted. SARC do not specify any events or actions for the movement of their player character. Lack of precision in describing events and actions is also common. The examples given in Table 14 illustrate that pupils do not conceptualise game play as it is

constructed because they do not have a clear mental model (Norman, 1983) of how a computer game is made. Several pupils still misunderstand what an event is or list general rules for the game as events; sometimes events and actions are confused or not separated.

If	Event (input)	Then	Action (output)
SARC If	Starman goes up a level	then	The enemies become harder to defeat.
AWTB If	You're in the air	then	Presses left/right they move in the air left/right.
CBMH If	Arrow keys or space pressed	then	Girl moves or jumps.
JBJG If	Horse eats	then	Munch sound.

Table 14: Lack of precision in referring to events and actions

Programming events

Whilst all pupils used events in their games with some success, some pupils found the concept of an event itself hard to grasp initially:

> JD: What are they doing on the event, what's the event?
> MB: What is the event? What does that even mean?
> JD: What are they doing there?
> JD: Miss? Miss we dunno what to do for the zombies on the event.
> MB: We know what action is but I don't understand the event.

Some of the difficulties pupils encountered with events arose because they assumed that the word 'event' had the same meaning in a programming context as it has in everyday usage. In the transcript LW says 'Ok, so the cheese is destroyed, so that's the event' - illustrating that she understood an 'event' as 'something which happens in the game', rather than as a trigger to make something happen (an input). MB refers to the *action* of his player character hiding behind a pillar as an *event*:

> MB: So for the next bit we have the man, and he … his … the event is him hiding behind … him hiding behind the pillars.

This misunderstanding of the specialist meaning in computing of everyday terms is identified in the literature as one of the difficulties of learning to program (see section

2.9). Confusion about events is also evident in cases where pupils discuss what events to assign to static objects which do not need events.

Whilst some pupils found the general concept of 'event' hard to understand, particular events caused others the following problems:

Collision event

One of the key interactions in the games pupils created is when two objects collide. Ten games made use of the *collision* event, although it was not always obvious to pupils which of the two objects should contain the *collision event*. At other times, *collision events* were added to both objects in the collision, which resulted in conflicting program code:

> **Obj_Control_link**
> **Collision event with door3:**
> Move to position (90, 50)
> Go to room room3 with transition effect Create from the top
>
> **Obj_Door 3**
> **Collision event with object Control_link:**
> Move to position (90, 50)
> Go to room room3 with transition effect Blend

Collisions also created problems when pupils (OWSW, CBMH) had set a parameter for moving objects as *solid*. If a collision is set up with a moving solid object, the collision checking routines of *Game Maker* fail and objects get 'stuck', as in the following example:

> OW: When the snowboarder crashed into a rock or tree, the tree or rock didn't disappear and went down the screen joined to our character.

Game Maker's collision detection also caused problems for two pairs because its default setting only registers a collision when there is a visible overlap between two sprites:

> AE: Well with the cars, for some reason, at first when you crashed into them they only, like, registered you had crashed when you got into the middle of the car, rather than when you actually hit it.

In JDMB's game the same problem occurred when the large surrounding canvas of two objects makes collision detection between them inaccurate.

Game start event

The *game start*/*game end* events also caused confusion. ACJC misunderstood the *game start* event, which determines what actions should occur when the game starts, and confused it with a command for the game to start:

> **Obj_Player character**
> **Other event: Game Start:**
> If left mouse button is pressed

Their intention was for the game to start when the left mouse button is pressed but there was no need for this event since they have used a start button on the title screen to start the game.

KW used the *game end* event to test the score, as the mechanism for advancing from level 1 to level 2 of her game. But since actions in the *game end* event only occur when the game has ended this cannot function as intended. This pupil confused the *game end* event with the *room end* event (end of the level):

> KW: I put 'game ends, if the score is larger than 500 go to next room', but it doesn't seem to work. You just carry on collecting ... until you've collected everything and then you just sit there and nothing happens.

Such errors occurred because pupils had not had sufficient practise in using these events and had not followed tutorials or received instruction in how some of the events function.

Duplicating events

Errors also occurred when pupils duplicated events, or assigned more than one function to a key. JDMB had problems controlling their missile objects because they duplicated *keyboard* events for the space bar - they used the space bar 4 times and referenced it in 3 different objects. The same pair also duplicated *keyboard* events for the left and right arrow keys in their player character and brick objects. This had the unintended effect of moving the brick and player objects together when those keys were pressed. They also duplicated instructions for the creation of a stone object and its upwards movement by putting similar code for this action in both the player character and stone objects:

> **Obj_Man**
> **Key Press event for <Space> key:**
> Create instance of object stone at relative position (0,0)

Obj_Stone
Key release event for <Space> key:
Create instance of object stone at relative position (0,0)

Create event:
Set the vertical speed to -3

Keyboard event for <Space> key:
Start moving in directions 000000010 with speed set to 2

As well as generating unnecessary code, such duplications introduced conflicts in the program and explain why the missiles do not behave as intended. In terms of program design, it would have improved the coherence of the gameplay if this pair had reused the same code for each of their three missiles, but at this stage in their learning they did not recognise that code segments can be reused to improve program efficiency.

These errors arose because pupils did not plan their game interactions systematically. Since they were programming 'on the fly' duplications and conflicting code were more likely to occur.

8.3.4 Actions

Planning actions

In their planning documents several pupils did not detail all the actions for an object - for example they forgot to add actions to play a sound when points are gained or to destroy an instance once it had been collected. Sometimes several actions were conflated, as shown in Table 15.

Object	Event (input)	Action (output)
SARC Starman	Click space bar.	Space bar makes Starman jump onto platforms to collect coins to give him points.

Table 15: Conflating actions

At other times, no actions were specified or actions were indicated, but not in sufficient detail; only 1/12 pairs specified the speed that an object should move; of those who added a score action some forgot to specify the value of the score.

These data illustrate that at this stage some pupils did not understand the need to decompose the gameplay into separate functions and to give precise instructions. This

inability to plan their programs in detail and to break their programs down into sub-programs is identified in the literature as one of the areas of difficulty that pupils encounter while learning to program (see section 2.9).

Implementing actions

Another source of difficulty for pupils was in selecting appropriate actions to achieve the object behaviour they intended. In *Game Maker*, each action symbol depicts a graphical representation of its functionality; a textual description appears on rollover. Despite this support, some pupils did not understand what action the symbol performed. Although they had been provided with a glossary of all the actions they did not make good use of this information:

> CB: I didn't know what any of [the actions] did or how to use them so I didn't know how they could help me improve our game.
> CJ: But if you hover over them they tell you what they do.
> CB: I didn't know what it meant.

The number of different actions was also a source of confusion for some:

> TB: All the controls are quite complicated, the amount of different things that you have to put in.

> CB: We have not finished doing all our actions and events yet because at first we found this very confusing.

> GW: I don't like it when you have to do the actions … And the events - that's really confusing.

Particular problems that pupils encountered with actions are indicated in the following examples.

Conflicting actions

Errors occurred when pupils had programmed conflicting actions, and this was the case in 6/12 games. In this example (ACJC) the first action instructs the ball to move downwards, but the second action instructs the ball to move to a random position:

Obj_ball
Create event:
Start moving in directions 010000000 with speed set to 2
Move to a random position with hor snap 0 and vert snap 608.

OWSW and JBJG make similar errors.

In another example LWGW give conflicting instructions in the 'down' *key press* event for their mouse object, where, in addition to the actions to move down, instructions are also given for it to move left, right and upwards:

Obj_Mouse
Key press event for <Down> key:
Start moving in directions 000100000 with speed set relative to 3
Start moving in directions 000000010 with speed set relative to 3
Start moving in directions 000001000 with speed set relative to 3
Start moving in directions 010000000 with speed set relative to 3

JDMB duplicate the *destroy instance* action for their monster object by putting it in both stone and monster objects:

Obj_Monster
Collision event with object stone:
Destroy the instance
Move relative to position (0,0).

Obj_Stone
Collision event with object monster:
Destroy the instance
Jump to the start position

Conflicts also arose when actions were duplicated in separate events for the same object, such as in this example (AWTB), where neither the event nor the action are necessary, since downward motion for the player character is effected by gravity, which they have specified in a previous event:

Obj_Player character
Keyboard event for <Down> key:
If relative position (0,3) is collision free for only solid objects
  Move relative to position (0,3).

LWGW give conflicting instructions for the movement of their cat object. In the first event an action instructs the cat to move randomly in all directions, including stop; in the second event an action instructs the cat to move randomly in all directions when it collides with a cheese object:

Obj_Cat
Create event:

Start moving in directions 111111111 with speed set to 4
Bounce not precisely against solid objects

Collision event with object obj_cheese1:
Start moving in directions 111101111 with speed set to 4

Other pupils select the wrong action. SARC's player character does not move because although they have set *keyboard* events for the arrow keys, no actions for movement are added:

Obj_Player character
Keyboard event for <Left> key:
Set the sprite to spr_starman with subimage 0 and speed 1

Keyboard event for <Up> key:
Set the sprite to spr_starman with subimage 0 and speed 1

Keyboard event for <Right> key:
Set the sprite to spr_starman with subimage 0 and speed 1

Keyboard event for <Down> key:
Set the sprite to spr_starman with subimage 0 and speed 1

The *change sprite* action used here simply changes the sprite and has been selected in error; because this action refers to speed, they thought it was a *move* action. It is clear from their code that they have not planned the game interactions and have not been able to apply what they have learned in the video tutorials to their own game.

8.3.5 Setting values/parameters/arguments

In their planning documents it was common for pupils to omit details of the parameters and arguments associated with actions. For example, several pairs omit to set speed for movement (KW, JDMB, JBLA, JBJG, AWTB, ACJC, CBMH) or do not specify direction (OWSW).

In the games themselves, some pupils did not select the correct values to achieve their intentions. For example, level 2 of AEMD's game presents a bird's eye view of cars driving along a street. The player's car has to overtake other cars and avoid collisions. However, because they have not applied the correct values for the *y*speed/direction setting of the non-player cars, these appear to be moving backwards.

In CBMH's game, they select a *set gravity* action for their spider object but then do not add a value for it, so no gravity effect is observed:

Obj_spider
Create event:

For all girl: start moving in the direction of position (0,0) with speed 0
Set the gravity to 0 in direction 0

OWSW attempted to use a conditional statement to control the appearance of their tree object - the structure is correct for these but the arguments are not. The intention is that when a tree disappears from the bottom of the screen in their vertically scrolling game, it should reappear from the top of the screen, but this does not function correctly because they selected the wrong value for the *y* coordinate - forgetting that *y*0 is located at the top of the screen:

Obj_tree
Create event:
Set the vertical speed to -4

Step event:
If y is larger than room_height
  Move to position (random(room_width),-65)

Self/other

Errors also occurred in the setting of certain parameters, such as whether an action following a *collision* event should apply to the object itself, or another object (JBJG, JDMB, LWGW). In this example pupils apply the *destroy instance* action to the wrong object in a *collision* event and create a collision between an object and itself:

Obj_log
Collision event with object obj_log:
At position (1,0) draw the number of lives with caption Lives:
Destroy the instance (self)
Move to a random position with hor snap 0 and vert snap 320

In the same game, an apple should disappear when a horse 'eats' it, but since the *destroy instance* action has been applied to the wrong object, the horse disappears:

> JB: Oh my gosh! No, we don't want the horse to disappear! No, it needs to delete itself! Oh blimey.

JDMB and LWGW also have problems in selecting the correct parameters to ensure that objects disappear as intended following a collision event.

Such examples show that when pupils do not plan object interactions before they implement their games, or where they do not check the logic of their code,

unnecessary errors occur. However, other problems may arise due to pupils' lack of familiarity with game programming concepts.

Solidity

In *Game Maker*, some objects need to be defined as 'solid', particularly if they are non-moving objects. Generally, moving objects should not be solid. This caused confusion because intuitively, pupils thought that all objects in a game would be solid.

Errors occurred in 6/12 games, when pupils selected the wrong value for this parameter. ACJC, CBMH and SARC made moving objects solid, when this was not required. In AWTB's game, the player character drops through the bottom of the screen, because the wall object used as a barrier was *not* defined as 'solid'. ACJC, LWGW, AEMD, and JBJG also failed to make some objects solid, when this was required.

Relative and absolute value

As noted in Chapter 7, pupils need to understand the concept of relative and absolute value in order to be able to manage variables effectively (e.g. the increase or decrease of their score, object position, speed). Errors were made with this aspect in 5 games. Some pairs did not make a value relative, when they needed to, for example CBMH's, JBJG's and OWSW's score did not increase relative to the current score, so it remained at an absolute value. Others made a value relative when this was not required: ACJC set the score relative to 0 at the start of the game; LWGW set the speed of their player character relative to 3, which had the effect of increasing the speed relative to the previous speed, making the object move increasingly quickly, which was not their intention.

Other errors occurred because pupils were encountering concepts for the first time and needed direct instruction to understand and implement the concept, as indicated in the following section.

8.3.6 Conditional statements

In *Game Maker*, conditional statements are achieved by selecting a *test* or *check* action and were commonly used in pupils' games to check if the score had reached a certain value or to check the position of objects on the screen. Although 6 of the 12 games included a conditional statement, the same number did not, which suggests that some pupils found this construct difficult to understand or to implement:

JB: We found [using the test variable action] hard. We did try to do that on ours but it wasn't working so I had to take it out.

MH: If she loses all three lives then they...
CB: So, block … Oh no, wait. We need to do the 'IF ...' um ... what shall we write?
MH: I don't know how to do it. I think you need to go over to the … I don't know.
CB: Me neither.
MH: Let's pause it and ask Miss.

Three games (ACJC, JBJG, OWSW) contain errors in the use of one or more conditional statements, commonly because no action is specified if the condition is met, as in the following example (JBJG):

Obj_Horse
Create event:
Draw the lives at (20,10) with sprite spr_horse
If lives are equal to 50

KW created a button in her title screen to show the high score table, but did not add a *test variable* action in the game to launch the high score table after a certain condition has been met (e.g. if all lives are lost). KW also used a conditional statement to test the score, but put this action in the *game end* event so it did not function correctly.

It is likely that the six pairs who did not use a conditional statement found the construct too difficult to implement without instruction.

8.3.7 Loops

As indicated in Chapter 7, some pupils made use of a loop construct to repeat an action or to continually check a game state, although not without difficulty. JDMB used the *step* event to repeatedly create new instances of their monster object, however, this caused the game to freeze because 30 monsters were created every second, which consumed too much computer memory.

JBJG attempted to use the *alarm* event, to repeat an action, but without success:

Obj_Player character
Alarm event for alarm 0:
Start moving in directions 000010000 with speed set to 0.

Here, the alarm event is incomplete - the alarm 0 has not previously been set, so there is no time interval to count down before the *alarm* event's actions execute.

Looping mechanisms such as this are not intuitive and pupils needed direct instruction from teacher, tutorials or peers to use them successfully.

8.3.8 Variables

Most pupils in this study made use of *Game Maker's* inbuilt variables to manage score, health and lives data. The following section describes specific problems which pupils encountered with these.

Score

Six pairs attempted to include some kind of scoring system, but most had problems with some aspect of it:

> JBr: Yeah my score stopped going up after a while. I think it was to do with going in and out of buildings and stuff.

> AE: We had problems with our score as well because on level 1, say you get 500 points, we found that when you go into level 2 it reset your score to 0 for some reason.

Those pairs who had implemented more than one level (ACJC, AEMD, JBLA) commonly had problems with managing the score across the different levels:

> AC: We've had a bit of bother with the levels … because every time … when we completed the first level the first level worked brilliantly, which was a big thing, but then after we got to the next level, 'cos we had an action saying 'if score is larger than 299 advance to the next level', whenever we got a Mars bar to give us some points it advances us to the next level straight away, which we didn't want it to do, we wanted it to [be added] to the score.

Because they had assigned the *test score* actions to the player character, which appears in levels 1, 2 and 3, the score thresholds for level 1 also take effect in level 2, as illustrated in the code below:

> **Obj_Player character**
> **Collision event with object obj_mars1:**

```
If score is equal to 280
  Display message: Congratulations! Advance to the next level!
  Go to next room with transition effect Blend
If score is equal to 580
  Display message: Congratulations! Advance to the next level!
  Go to next room with transition effect Blend
If score is equal to 980
  Display message: You are the Ultimate Dodgeball Champion! Well done dude!
  Show the highscore table
  End the game
```

Others had problems with programming the score to increase or decrease correctly. For example, in CBMH's game the score had not been set to decrease relative to its current value when points were lost, so the score remains at an absolute value of -10; JBJG's score does not increase relative to its previous value. JDMB set the score variable at the start of their game but there is no additional code to increase or decrease the score. In KW's game, the score mechanic functions correctly on increase but there is no mechanism to lose points. OWSW program the score to decrease correctly, but it does not increase in relative increments, even though in the voice data, OW shows he understands the need for this.

Sometimes score mechanisms were attempted but incomplete. In JBLA's game there are no collectable items and no interactions to generate a score in the first two rooms of the game. The score mechanic only activates when the player character enters room 3, but does not function correctly because the code for increasing the score is placed in the *create event* of the collectable item instead of in a *collision event* between the player character and the collectable items. In JBJG's game, the score increments correctly when a carrot object is collected, but since no instances of this object have been placed in the room, the score associated with it cannot function and nor is it displayed on the screen. Level progression is achieved in KW's game by attaining a certain score, but since the *test score* action is placed in the wrong event (*game end*), this condition can't be met and the player cannot progress to level 2. A high score table is partially implemented, but no condition has been set to make it appear.

Five games do not include a score mechanic (LWGW, SARC, AWTB, CBMH, GS), which suggests that some pairs found this aspect of game authoring difficult, less important than other game features, or they did not have time to implement it.

Some of the errors which pupils encountered with the score mechanic, such as remembering to make the score increase and decrease relative to the current value, could have been corrected if they had checked their program code. Other errors arose

because pupils had not learned how to manage score effectively across levels, suggesting that this needs to be modelled by the teacher or a tutorial. It is unlikely, without direct instruction, that pupils will understand how score is stored as a variable in a game.

Lives

Four pairs attempted to add a variable to store lives data in their games, but 3/4 of them had problems with displaying the lives status:

> JC: Me and A. couldn't get the lives to appear on the screen for some reason.

> JB: We were trying to get the lives up on the screen and they weren't coming up at all.

Pupils expected lives status to appear on the screen when they used the *set lives* action, in the same way that the score is automatically displayed in the game window when a *set score* action is used. Pupils did not realise that to display lives on the screen as text or a graphic, they had to use a *draw event* and a *draw lives* action. ACJC set lives, but did not add a *draw event* and *draw lives* action, so the lives status does not display; JBJG and OWSW did not set a value for lives at the start of their game, but used an incomplete *test lives* action instead, so the lives mechanic does not function correctly. JBJG included a *draw lives* action but did not assign it to a *draw event*, so their lives graphic does not display on the screen. Since only 1/12 pairs managed to correctly display a lives graphic, (and they followed a tutorial) this suggests the need for more formal instruction in the use of the lives mechanic. Similarly, of the two pairs who attempted to include a 'health' status bar (AEMD, JBLA), only one managed to display this on screen.

Adding /losing lives

Further problems were encountered in the mechanisms pupils used to add or lose lives. In JBJG's game lives are awarded when the player character, a horse, eats an apple. A *set lives* action is correctly placed in a *collision event* between the two objects, but because there is no code to display lives, the lives status is not visible on screen. It was also difficult for the horse to 'reach' some of the apples, which appear randomly in the room, and once eaten they do not reappear, so after five seconds of game play the apples disappear and there is no mechanism for increasing the lives value.

KW had similar problems: lives are set at the start of her game but no code is given to display the lives on screen and no mechanism to lose lives is implemented. In level 2 of AEMD's game, the lives correctly decrease when the player character collides with the non-player character, but since the lives caption has not been implemented, the lives status does not display on the screen.

Pupils did not attempt to program a lives mechanic in seven games (LWGW, AWTB, CBMH, GS, JBLA, JDMB, SARC). The data here suggest that pupils are unlikely to learn how to implement score, lives and health variables by experimentation and exploration alone. While *Game Maker's* 'Help' menu and two of the tutorials made available do include some information about variables, pupils were generally unwilling to read text tutorials or to follow more than one video tutorial. This meant that they did not access some of the information they needed regarding the use of these variables.

8.3.9 Miscellaneous errors

Many of the errors observed arose because pupils did not plan their game interactions systematically or check the logic of the code they later implemented.

In AEMD's game, the win state in level 3 occurs when all instances of an object (obj_key) are collected, but does not function as intended, because the two separate segments of code which refer to this object cancel each other out:

Obj_Key
Collision event with object obj_player:
Play sound snd_key; looping: false
Change the instance into object obj_key--x, not performing events

Obj_Controller key
Step event:
If number of objects obj_key is equal to 0
  For all obj_door: destroy the instance

The code in the *step* event 'destroys' (opens) the doors if all the keys are collected (i.e. if the number of keys = 0). But in the *collision* event, when a key is collected, it is programmed to change to an invisible key object. In effect, the key object being counted in the *step* event no longer exists because it has been changed into another object - there are no original keys for the condition to count. When all keys are collected the win state is not triggered because the controller key object was not placed in the room.

Sometimes simple errors occur because code is incomplete. LWGW programmed their cheese objects to disappear when the mouse object collides with cheeses1, 2 and 3 - but instances of other cheese objects placed in the room (i.e. cheeses 4, 5, 6) do not disappear, because no code has been written for a *collision* event with them.

Another code section instructs cheese objects to stop moving after they have been destroyed. Clearly this cannot have been what they intended. It is not appropriate to select a *move* action for a stationary object and no events or actions are required for these cheese objects since a *destroy instance* action for the cheese has already been set in the mouse object (so that the cheese disappears when the mouse 'eats' it).

In JBLA's game an error is made when a *collision* event between the player character and another object is set for an object which they have placed in the credits screen (i.e. outside of the game room itself), so the main player character cannot 'collide' with this object and the resulting action cannot occur as intended.

Since reading and understanding code is an important part of learning to program (CAS, 2012a), pupils need to be encouraged to view the textual information for their objects, and check for obvious errors such as these.

8.4 Summary

This chapter has summarised the difficulties which pupils referred to in the voice data and interview transcripts, and the errors identified in their planning documents and in the code of their authored games. In documenting these problems, it becomes clear that game programming is conceptually and practically challenging for pupils, not least because the activity introduces such a wide range of concepts and practices. Learning which programming constructs are required to create particular effects is the most difficult aspect (Cheng, 2009; Macklin and Sharp, 2012) and the findings presented in this chapter suggest that this knowledge and understanding cannot be acquired efficiently by all pupils without some form of instruction, delivered either by tutorials (video and/or print) or teacher intervention (modelling, structured activities).

It is also important to note that the problems pupils encountered were, to some extent, related to the amount of time pupils had been involved in the project (16 hours). Although the time given to the game authoring activity was greater than for other

projects they had completed, they were only at the beginning stage of learning to program.

Learning how to use the software well enough to create a satisfying game, and learning how to implement key game mechanics adds to the complexity, yet pupils were reluctant to follow more than one video tutorial and resisted reading printed guides or watching teacher demonstrations which would have helped them to program their games more effectively.

In their eagerness to begin making their games, some pupils did not plan their game interactions effectively and this caused problems at the implementation stage. For some pupils, using a ‘bottom up’ approach, or working without a plan, was not a successful strategy. Although pupils express a preference for working directly in the software and learning by doing (see Chapter 5), these findings suggest that accuracy and efficiency are compromised when they do so.

Because they were programming ‘on the fly’, rather than systematically, their thinking was sometimes muddled and resulted in avoidable errors. They also did not approach debugging systematically and failed to check their code for obvious errors, suggesting that these practices need to be more strongly modelled and integrated into schemes of work.

These practices of reading and checking code, program planning, and adopting more systematic approaches to dealing with errors are identified in the literature as important for learning to program (see section 2.9). In fact Perkins et al. (1986) suggest that encouraging better learning practices such as these is central - and this echoes one of the ‘eight big ideas of constructionism’ - children need to learn how to learn (Papert, 1999a).

Critics of constructionist approaches to teaching programming claim that these are not well aligned to the domain (Ben-Ari, 2001; Meerbaum-Salant et al., 2011) and suggest that exploratory learning needs to be supplemented by planning and formal methods. Others observe that constructionist approaches may not be well suited to the early stages of learning to program for some learners (Guzdial, 2009) and that while some elements of programming can be learned with minimal teacher input, for more complex programming constructs teacher intervention is required (Murnane, 2010; Denner et al.,

2012). The findings presented in this chapter support this position and the implications of this are discussed in Chapter 10.

Chapter 9 Affective values of authoring computer games

9.1 Introduction

This chapter considers the value of computer game authoring in the affective domain, an important consideration in constructionist learning theory (Papert, 1986; Kafai and Resnick, 1996b). As noted in Chapter 2, authoring computer games is widely found to be motivating (see section 2.4) and to give rise to positive attitudes to learning. At the same time Papert identifies 'hard fun' as one of the 8 'big ideas' of constructionism (see Chapter 3), and argues that children are more likely to enjoy what they do when the activities they engage in are challenging. This chapter presents pupils' perceptions of the affective values of the game authoring activity, as evidenced in their digital voice recordings, journal entries, and interview data. These findings add further support to previous studies of children making computer games, but extend and update them by focusing on a different context - the use of *Game Maker* in the UK secondary ICT curriculum.

9.2 Enjoyment and engagement

The unit of work was an implementation of a constructionist learning activity, characterised by its collaborative work pattern, extended time frame and personally and culturally meaningful outcomes. Pupils in this study frequently expressed feelings of having enjoyed making a computer game - and this finding is widely supported in the literature (see Kafai, 2001; Kafai, 2006b; Robertson and Howells, 2008; Cheng, 2009; Carbonaro et al., 2010; Li, 2010; Baytak et al., 2011). That they enjoyed the activity is evident in the enthusiasm shown by some pupils:

> AC: When we are creating a game we come in and sit down and we get on with our game straight away.
>
> GS: I can't wait until I have finished my game!
>
> MH: C. and I both look forward to start to actually make the game and cannot wait to see what it will look like when it is finished.

This section considers factors which gave rise to enjoyment and engagement, beyond the collaborative, constructionist working pattern, which is discussed in Chapter 5.

9.2.1 Fun

Notions of 'play' and 'fun' are brought to the fore in this type of activity and this is a probable factor in pupils' enjoyment of it. Much of the work pupils were engaged in involved playing their games and of course they were creating their games for others to play. The word 'fun' appeared in a third of pupils' journal entries, a fifth of the digital voice recordings and in all of the group and pair interviews. Just over half of the pupils (12/22) used the word at least once to describe some aspect of their experience of making a computer game (7/12 boys; 5/10 girls).

The activity itself was perceived by some to be more 'fun' than other types of work they had encountered:

> MH: I think … it's fun as well, like when you say we're gonna make a game, everyone suddenly goes 'Oooh' … rather than spreadsheets – it's funner [sic].
>
> AE: 'Cos some things like spreadsheets are challenging but some people don't seem to think they're fun, but with a game most people think they are fun.
>
> JD: And you can, like, make a story, not do like a boring spreadsheet, you're just making a story up, so it's fun for you too.
>
> AW: It's a good project and it's fun to make, 'cos you're creating a game which you can then play, so it's learning new skills which are fun.
>
> MD: Yeah [the animation] was fun. I hadn't really done something like that before.

Other references to fun were made in general comments about the activity:

> AE: It was challenging but also fun at the same time.
>
> GS: Although my game is kind of hard to make ... I find it much fun too.
>
> JB: It was fun. I wish we had had more time so we could finish it though.

KW: Yeah I found it fun.

JG: It's fun to make, just ... all muddled up.

This finding is consistent with other studies, where students used the word 'fun' to describe the experience of creating games (e.g. Li, 2010; Navarrete and Minnigerode, 2013; Yang and Chang, 2013).

One aspect of the activity which may have contributed to pupils' experience of fun was that the mode of learning they were engaged in was playful and experimental. Exploratory learning is one of the cornerstones of constructionist philosophy, and most pupils in this study enjoyed this approach. In programming their games, they tried things out to see what would happen and these actions were provisional. As such making a game was a supportive learning activity. Pupils had an almost immediate visual feedback of what worked and what didn't work on screen and could attempt to solve problems by refining parts of their programs step by step. Thus they came to view errors as problems to solve:

OW: When we made our snowboarder move, first of all he went out of the screen so we had to make an action to keep him inside the screen. Then we also had a problem when the snowboarder crashed into a rock or tree, the tree or rock didn't disappear and went down the screen joined to our character. So we made our character jump to a given position on the screen when he collided with an object. Our latest problem is that the objects don't reappear when they go out the screen so we will create an action and event that says 'if object is larger than room height, reappear at random position inside the room'.

Papert argues that this 'natural', 'Piagetian' learning should be given more status in schools (Papert, 1980b) and 'playful pedagogy' also has more recent support (e.g. Morgan and Kennewell, 2005; Kennewell and Morgan, 2006). Additionally, learning that 'getting things wrong' is part of the process of programming changes pupils' attitudes to making mistakes. Having the 'freedom to get things wrong' is an important feature of constructionist learning theory (Papert, 1999a) and this different approach may have been a contributory factor in some pupils' enjoyment of the game authoring activity.

9.2.2 'Hard fun'

Whilst most pupils agreed that the game making experience was fun, they also acknowledged that it was a difficult project (see Chapters 6 and 8 for a discussion of difficulties pupils encountered). It was 'hard fun'[6] (Papert, 1996a). The data show that the project gave pupils an appreciation of the complexities of creating a computer game:

> AE: I learned how hard it was to create a game. It was especially hard to think of the rules and to work out how to apply them using the events and actions. [But] I really enjoyed the game making project and I learnt a lot.
>
> AW: We've learned how to make and program a game and that it's not actually simple to do, it takes quite a while 'cos you need to plan the game and what you want to happen in it.
>
> MH: When you think of making a game you think it's going to be really simple and really easy, but it's not, because [there are lots of] different topics which you have to cover.
>
> JB: It's harder than you think, 'cos you sit there and you're playing a game and you're saying 'this game is really easy to play so it must have been easy to make', but then when you come to actually make the game yourself you find out how hard it is.
>
> MH: Yes that was what I was worried about with our game, it was really hard to make, but I reckon that it would be really, really easy to play.

Some pupils found the game making activity more difficult than other tasks they had completed earlier in the year:

> JB: It makes you think more than the plain work that everybody does … like simple Microsoft work …This one makes you think really hard about what you're doing.

[6] Papert coined this phrase to defend the challenging nature of computer-based constructionist activities, arguing that, "fun and enjoying doesn't mean 'easy'" (Papert, 1999a).

> AW: It's harder than the other work and it's something we that haven't done yet or used so it's a new skill that we've learned.
>
> JG: I liked it, but I think it made you think too much.

This experience of game making being a difficult task, while also fun, is supported in the literature (e.g. Kafai, 1996; Li, 2010). Papert's view that children prefer challenging activities as long as they are also interesting (Papert, 1998b), and personally and culturally relevant (Papert, 2002) is corroborated by these findings.

Giving pupils challenging tasks which stretch them yields positive affective gains too. In the current study, several pupils expressed pride in their achievements, because they recognised that they had found some success with a difficult task:

> CB: It wasn't easy, but I thought it would be a lot harder, like impossible and it wasn't. Yeah, although the games are quite basic they are good.
>
> MD: I knew we could make a game but I didn't think we could make one as sophisticated as we have made.
>
> JB: It was hard in its own way to get it to, like, do things and it was a good achievement.

Others seemed to value the challenge inherent in the activity, which arose out of having to integrate different skills and simultaneously consider different aspects of the game authoring process:

> MD: I think when you're making a game you have to tie in everything you've learned, 'cos you have to include graphics, you have to include what the player's going to think when they're playing it and you have to sort of communicate with the player and you have to think about ... you have to do a bit of maths and stuff. It uses a lot of different skills.

These findings extend those of other studies using different software (*Alice*, *Scratch*) which also document pupils' sense of achievement in creating a computer game which they acknowledge to be a difficult task (see Ferdig and Boyer, 2007; Werner et al., 2009; Li, 2010; McInerney, 2010; Navarrete and Minnegerode, 2013).

For some pupils, engaging in difficult tasks over an extended timescale fostered perseverance and commitment, as well as other positive learning behaviours, such as those promoted in guidance for developing effective learners (DfES, 2004) and the personal, learning and thinking skills framework (QCA, 2007a):

> MH: C. and I took great time deciding on our game ideas. We planned and thought it through, and even when we thought we had a good story line, we decided to re-think and make it more suitable for Year 6 pupils.
>
> JB: Yeah we couldn't get the horse to jump. We tried for, like, a whole lesson to do that.
>
> LW: My game was successful in that I achieved the basics of my game that I was determined to do, although I did not completely finish and had many problems that I could not fix in the time I had.

Even though some pupils experienced many problems, they did not give up but worked hard to find solutions. Because the problems were directly linked to games, and therefore authentic, there was a purpose in trying to overcome them, even if that meant simplifying or modifying an idea:

> TB: We're considering, instead of [scrolling horizontally], [scrolling vertically] and then having a finish at the top because it's a lot easier to make, a lot, lot easier and we'd be done probably by the end of this lesson maybe.
>
> LW: We had trouble with our background and sprite sizing which slowed us down. We got round this by making the background larger by going back into [*Fireworks*] and making the channels larger and the actual cupboards smaller. Then we exported the background to *Game Maker 7* and placed in backgrounds.

Developing learners' perseverance is one of the positive outcomes of this project - and this persistence and commitment to making their games is a consistent finding in related research using different software (Kafai, 1996; Howland et al., 1997; Robertson and Howells, 2008; Cheng, 2009; Li, 2010) and, it is suggested, develops in this context because pupils have been given projects with sufficient duration to enable such learning skills to grow (see Harel, 1991; Kafai, 1995).

9.2.3 Interaction with the software

Another factor which contributed to some pupils' engagement was the interaction with the software itself and they described their use of *Game Maker* and the video tutorials as fun:

> CB: I had never used software like this before so it was a fun, new experience.

> MH: I enjoyed using the *Game Maker* software as well as I had never used it before and it was good fun to try it out.

The fact that the software was new to them seems also to be a contributory factor to their enjoyment of it - not only did it give them a different experience, but it contributed to their sense of competence when they were able to use their prior learning of other software to help them manipulate new interfaces.

For some pupils, engagement with the software seems also to be due to the imaginative purposes which it was used for. Because pupils were using the software as an expressive medium to create a game they enjoyed using it. This contrasts with the comments they made about working with spreadsheets, for example:

> GW: [A game is] like a good end product … like I am interested in, rather than like a spreadsheet.

> MD: With a spreadsheet you're sort of working to a ... you know exactly what ... you make a spreadsheet because you'd have a goal in the first place, like you're making a spreadsheet 'cos you want to solve a problem, whereas with a game you're making something just to entertain, so it's completely open ended.

This 'open-endedness' was valued by some pupils and they associated it with being given freedom, which they also valued. Open-ended tasks also added challenge:

> MD: In *Game Maker* you have more freedom to challenge yourself, 'cos if you're doing a database, you only need to do what the database is designed to do, someone just tells you what they want it to do and you make it, whereas with a game you can make a game as complex as you like.

> AW: In *Flowol*, all you have to do on there is just make a circuit which you can sort out, which is just a circuit, it's a loop back that you send around, but with *Game Maker* you *create* that circuit, but you need to make a whole lot more to actually make it work.

The relationship that is set up between the pupil and the software also seems to lead to engagement, arising out of what Zorn refers to as a "continuous interaction of actorship on the part of the designer and the technology as actant" (Zorn, 2009: 361). The software was responsive and this gave pupils the feeling that they had some agency. Making something work was rewarding. Moreover, these interactions led to engagement because they took many forms - pupils worked on a variety of activities: creating an animated splash screen, adding a high score table, creating animated graphics, programming objects, experimenting with the actions and associated parameters, and testing their games.

9.2.4 Creativity

Indeed, several pupils valued the creative aspect of the game-making task and particularly enjoyed being given what they regarded as creative freedom:

> AE: You can be quite creative 'cos there's loads of stuff you can do with *Game Maker*. You can make *any* game.

> KW: It gives you more independence and you get to choose what you're going to do and be more creative.

> TB: And you can be more creative with this. You can do what ... yeah you have your own choice and opinion on what to do, 'cos with *Flowol* you have to do what it asks you to do, otherwise it won't work at all. You have options of making what you like in *Game Maker* … And you can be so creative with it ... it's brilliant!

> JBr: It expands your creativity 'cos you need to think about things more, like ideas and stuff, and you are working harder than you'd usually do, if you want to get stuff done.

> AC: I said earlier that you think that a game would be quite simple to make but actually the creative mind behind it is quite vast.

Although the games pupils made were quite derivative, they still felt as if they were being expressive and original - in that their game was different to anyone else's in the class, and a new mode of creative expression for them.

Some pupils valued the game authoring activity because it was a different activity for them and they agreed that making a real product which others could play set this type of work apart from other projects they had done in ICT and elsewhere in the curriculum. In fact, having created a 'proper' product was valued because it increased the status of their work in their own eyes:

> MD: I like the intro screen … it's really cool. I think it makes the game look really good, makes us look professional.

They had made something that "can be shown, discussed, examined, probed and admired" (Papert, 1993: 142) and they seemed to value this:

> MH: You're saying it's like English, Maths and everything, but I dunno. It's kind of a completely different, like, topic for your brain as well, cos it's not something you do normally … Just everything about the topic was new to me.

> CB: In, like, Year 7 and Year 8 and Year 9, before we did game-making, we did stuff on spreadsheets several times, yeah we did similar sort of stuff but then game-making was, like, completely nothing like we'd ever done.

> SA: When I thought of ICT before the game-making topic I always thought of spreadsheets and things like that.

This idea of difference is also conveyed by some pupils, who felt that the games they created were individual and unique, and this echoes their comments about how they like to be able to work as individuals and not as a whole class:

> JB: Not everyone was doing exactly the same work. Well we were doing the same, but different sometimes.

> CB: Yeah, like if you had a class's, like 30 pupils' presentations they'd all be like quite similar, but then if you had 30 people's games they would all be completely different.

> AC: I think *Game Maker* allows you more freedom because in our last topic about the 'Grease' thing, it was sort of set, we had to do this, we had to do this, we had to do this. There was a bit of freedom in the differences in prices and blah, blah, whereas with *Game Maker*, everybody is doing something different, there's a totally different storyline for everyone.

These comments suggest that pupils value activities where they can express themselves as individuals, in a creative task - so that the outcome is personally meaningful to them, not just an individual response to a uniform task. In this way the game-making task was perceived by some to give them more creative freedom. These findings validate constructionist theory in its assertion that learners are more likely to become intellectually engaged when they are working on personally meaningful projects (Kafai and Resnick, 1996b: 2).

9.3 Preparation for work

Whilst a significant number of the pupils in this study referred to game making as 'fun', almost one third (7/22) of pupils saw value in the game authoring activity because of its relevance to the games industry and as a preparation for possible future careers:

> SA: It was a good experience if we ever want to do something similar in later life.

> MD: Well I think it can be useful. Well I mean games are a very large market aren't they? There are so many games so to know about games is useful 'cos it could be a future thing … if you want to be a game authorer [sic].

> TB: There's such a wide industry in games at the moment, with all children playing games all the time, that ... it's quite useful to know how to make them and how it works.

> AW: It teaches you important skills that you need for programming and if you wanted, it gives you an idea of what the job would be like, so yes it is important to learn.

AC: Well if you want to pursue IT as a career then you can always create games and *Game Maker* is just like a leverage.

TB: And it's not only [relevant] to game making as well, it can teach you about sorts of programming for other things as well maybe, so it can be used for more than just game making.

MH: Yeah I think it makes you realise that there's more to a computing career as well than just designing spreadsheets and stuff.

Although the game authoring activity had not been introduced to prepare pupils for the world of work, it is certainly of value that some pupils saw a purpose in and an application for their learning in terms of their futures. In the same vein, those who were involved in the recent drive to replace ICT with Computing widely refer to the needs of industry in their rationale (see Livingstone and Hope, 2011; Schmidt, 2011; Furber, 2012; Gove, 2012a). In this respect, the game authoring activity holds value since it presaged later exhortations to use video games in schools to draw greater numbers of young people into STEM and computer science (Livingstone and Hope, 2011: 82). It also belies criticisms that "the way in which ICT is taught in schools is failing to inspire young people about the creative potential of ICT and the range of IT-related careers open to them" (Gove, 2012a: n. pag.).

9.4 Different relationship with technology

From the researcher's perspective there is also value in authoring computer games in so far as it sets up a different relationship between pupils and technology, as illustrated in the following exchanges:

AC: Before we didn't know how to make a game and now we have got some idea of [how] a program ... will help us.

LW: When you play games now you see [it in a different way].
JD: You think about it how it's created and that.
SA: Yeah 'cos when you play a game you just take it for granted, really, as something that just ...
MD: Yeah ... works.

These comments are important because they show that such activities have the potential to make learners reflect on their own relationship with technology (Zorn, 2009: 341). In creating a game, the relationship between pupils and technology changes from being a consumer, to being a producer - and the educational, social and cultural significance of this is reported widely in the literature (Papert, 1993; Howland et al., 1997; Kafai, 2001; Turkle, 2003; Good and Robertson, 2004; Habgood, 2006; Kafai, 2006b; Pelletier, 2007; Hayes and Games, 2008; Robertson and Howells, 2008; Williamson, 2009; Games, 2010; Harel Caperton, 2010; Robertson, 2012).

Although the practice of 'designing' systems (presentations, websites, information systems, for example) was implicit in the programme of study for ICT in operation up until 2012 (QCA, 2007b), for some, their identification as producers was more strongly felt when making a computer game than in other units of work, and this may also account for why they valued the activity:

> MH: I just like the whole idea that we actually managed to *make* something that people can *play*.
>
> JG: Yeah, because at the end of it you can *play* your game and you think 'Oh I've *made* this', when the other [project] was a bit like, 'I've just made that' and you can't really play it and make it do things ... with this you can.

Their enthusiasm for the games they created gives support to the constructionist idea that pupils become more engaged in their learning when they create things which have personal meaning for them.

What is also important here is that pupils developed 'construction-oriented' relations with technology and came to understand that it is not only professional developers who can have agency over technology (Papert, 1993). They learned that game authoring is not something that others do - but something that they themselves can do, and this is certainly of value in terms of "identity formation and the construction of the literate self" (Jewitt, 2008: 46):

> JB: Um, yeah, I feel [it's] more of an achievement 'cos we have our proper finished, like, product ... it's not just a piece of paper, it's like *more*, and it's ... I feel really happy with it that we've managed to do something like that.

> MH: I thought it took professionals and professional software [to make a computer game].

Creating a computer game also changes the way young people relate to technology in that it gives them a context in which to learn 'about and with technology' (Kafai et al., 1997: 122) and this makes them more aware of the possibilities and constraints of technology. This conviction that children need to understand more about the technologies that pervade their world is strongly articulated in constructionist literature (see Papert, 1993) and has been reasserted recently (see Furber, 2012; Noss et al., 2012) by those who argued for the return of computer science in schools. The key idea here is that children should not just be passive consumers of opaque and mysterious technology (Furber, 2012) with no understanding of how or why the systems they use work (Noss et al., 2012). In making their games, some pupils in this study began to build such an understanding:

> SA: I didn't even know you *could* make a game. I've never had any experience of that ever.

> CB: I will probably think about it when I play on games in the future. I'll think, like, 'I wonder how hard it will be to make something like that' or 'I did something like that'.

> JB: I used to think computer games, why [did] they used to take so long to come out and now I know it's 'cos every single room needs to be modified and, like, every little bit in there needs to have, like, loads of complicated things just to do that.

Relationships with technology also changed because in using *Game Maker's* visual programming environment, pupils experienced a direct form of interaction with the computer (Cutts et al., 2011) and this is of value because it gave them a "[visceral] understanding of what a computer is and how one can interact with it" (Cutts et al., 2011: 137). Computer processes became somewhat demystified and pupils began to see that computers do precisely what they are told to do and no more, and conversely, have to be told precisely what to do in great detail. This was important learning.

9.5 New identities

All pupils adopted new roles as 'game makers', the significance of which has been explained above in terms of pupils' relationships with technology. But some pupils also developed new identities within the classroom. For example, three pupils (AE, MD, JBr) found themselves to be relative 'experts' in making games and this new-found expertise altered how other pupils viewed and responded to them. Their status in the group is likely to have been strengthened, since expertise in this context was recognised and valued by others in the class, who respected the boys' domain knowledge. These pupils, who were normally reticent in their contributions and interactions with others, now felt able to take on a more prominent role in the classroom. Two very able boys (AE, MD) in particular were approached frequently for help and this is likely to have had a positive impact on their self-esteem. Less socially confident boys like AE, MD and JBr were more able to speak to others because they had a reason and requests to do so. This finding is consistent with the outcomes of other studies using different software (see DEECD, 2010; Fowler and Cusack, 2011; Passey, 2012), where teachers reported that students who were generally quiet in class blossomed and adopted more prominent classroom roles when making computer games.

Making a computer game contributed to a new sense of identity for those who struggled with the activity too, in that it gave such pupils a sense of competence, a sense of belonging to a community of practice in which they could operate, at least on some level:

> SA: I feel now I could talk to somebody who knew more about it, like J. or A. for example.

The literature on situated learning (Lave and Wenger, 1991) and social learning in communities of practice is relevant here (Wenger, 1999). The idea of 'situatedness' is important to constructionism, in terms of promoting the development of learning communities and collaboration within them, but also in the sense that what children learn when engaged in constructionist activities is situated in 'computational microworlds' (Harel and Papert, 1991b) i.e. the programming environments which are used. The idea here is that working with computers enables pupils to "bring together *in their thinking* mutually supportive internal microworlds" (Harel and Papert, 1991b: 27). In terms of the current study, this means that pupils' new identities as producers of

computer games, and as learners, are situated in their interactions with the software, as well as in the surrounding classroom.

9.5.1 New identities as learners

New identities were also formed because pupils were able to bring into the classroom their out of school experiences with playing games and this may have given them a sense that they had something to contribute, that they knew something about this subject - and quite possibly more than their teacher. Thus a different relationship was set up between them and their work, because the work was to make something they had some knowledge about. All pupils in the current study had played computer games and accordingly, had some personal identification with and some expertise to bring to bear on the learning activity, which is perhaps less the case with other school-based IT tasks (such as creating spreadsheets or using databases):

> MH: I think it was good 'cos people play games in their spare time but you don't go home and make spreadsheets ... well I don't. It's something that people can relate to in everyday life as well, which is good.

Field notes for the pilot and main research record that some pupils (JBr, AEMD) became increasingly confident and more vocal when they were making games because they could express their interest and expertise in playing games at home, in the classroom setting. Those who used games they played at home as references for the games they made perhaps felt 'on home ground' (GS, JBLA):

> GS: I got most of my ideas from Pokémon Ranger Shadows of Almia. I can't wait until I have finished my game because I know if I put enough effort into it, I can make an equally good game as I do on RPG Maker [at home].

These findings illustrate the importance of harnessing and validating pupils' out of school learning as reported in the literature (e.g. Hague and Williamson, 2009; Madill and Sanford, 2009; Grant, 2010; Beavis, 2013) and authoring computer games emerges as an important context for this.

A different relationship was set up between them and their work also in the sense that they had created something of cultural significance, which had meaning for them. Some pupils particularly valued the fact that the outcome of their work was tangible, playable, and relevant:

JB: I feel like [it's] more of an achievement, 'cos we have our proper finished, like, product ... it's not just a piece of paper, it's like *more*.

AC: If you write an essay then basically the end product is a grade and you get your essay back, but with this you also get a grade but you get your game completed.

MH: I just like the whole idea that we actually managed to *make* something that people can *play*.

JG: Yeah, because at the end of it you can play your game and you think 'Oh I've made this', when the other [work] … you can't really play it and make it do things.

These findings give support to other studies (Sanford and Madill, 2007a; Cheng, 2009; Brennan, 2013a), where game making is similarly seen to validate young people's experience and their recreational lives.

9.6 Summary

In current UK debates about the ICT/Computing curriculum, there is a need to find a suitable metric for assessing the value of computer-based activities, since they do not all share equivalence (Stager, 2008). In terms of constructionist philosophy, activities such as authoring a computer game, which promote 'transformational' use of technology (ibid.) and which allow learners to encounter 'powerful ideas' (Papert, 1980b) are preferred.

This chapter has considered the value of authoring computer games from pupils' and the researcher's perspectives, and suggests that there is merit in the activity, which goes beyond the learning of curriculum content. This includes improved attitudes to and enjoyment of learning, as well as issues of identity formation and changing relationships with technology. The next chapter discusses the implications of the findings in Chapters 5-9 and concludes this study.

Chapter 10 Discussion and conclusions

10.1 Introduction

This study has explored the introduction of a unit of work in computer game authoring as part of the formal UK ICT curriculum for Year 9 pupils. In so doing it seeks to achieve an understanding of how authoring games using *Game Maker* supports the learning of basic game design and programming concepts and practices, what difficulties pupils encountered and whether the learning theory of constructionism provides a suitable approach for such activities.

Principal drivers for the research are a recognition of the importance of giving pupils opportunities to become producers as well as consumers of digital media (see Papert, 1993; Luckin et al., 2012; Nesta, 2012; Beckett, 2013; Sefton-Green, 2013) and the need to develop accessible introductory schemes of work to implement aspects of the Key Stage 3 programme of study (QCA, 2007b; DfE, 2013c).

The following section discusses the conclusions drawn from Chapters 5-9 and frames them in terms of the research questions they refer to.

10.2 Making games – the process

Chapter 5 addressed the research question 'What are pupils' perceptions of the process they followed during a constructionist designed game authoring activity?' The scheme of work had been organised following constructionist learning theory and pupils' responses to this approach provide insights into the practicalities of this mode of learning. Whilst other studies cited in the literature review draw on constructionism as their rationale for selecting game authoring as a context for learning (e.g. Baytak and Land, 2010; Vos et al., 2011; Denner et al., 2012), this study considers the constructionist process they followed, an area less focused on elsewhere.

The following sections consider the findings from Chapter 5 in light of the 8 big ideas outlined in Chapter 3, which constitute the theoretical framework of this research.

10.2.1 Learning by doing – big idea no.1

Whilst other studies promote learning by doing in the context of making computer games, they also acknowledge problems with this approach (e.g. DEECD, 2010; Baytak and Land, 2011; Doran et al., 2012; Bermingham, 2013; Ke, 2014). The current study extends these findings by offering useful insights into some of the drawbacks of this way of working. In particular, it reports that attempts to deliver elements of the unit of work in a semi-structured format were resisted by pupils, who preferred to work in an iterative, problem-driven way. Once underway, they sidelined planning tasks and were reluctant to follow teacher-led interventions. They wanted to make their games and 'do the learning' in *Game Maker*. This gives a clear indication that pupils enjoyed the constructionist imperative of 'learning by doing' (Papert, 1999a). However, in practice, this enthusiasm for active learning meant that important stages in the design process were avoided or not fully completed, which led to problems with the later implementation of some games.

Secondly, although constructionism promotes exploratory learning, where 'tinkering' and 'bricolage' are validated as alternative ways of working (Papert, 1980b; Turkle and Papert, 1990), there was a mixed response to this approach from the pupils in this study: some favoured this approach, while others found it less supportive (CB, KW). Some pupils (AEMD) were 'planners' and took a methodical approach to their work, others (SARC) were 'bricoleurs', less structured in their approach, and it is of significance that their games were less successful. Although pupils claimed to prefer the freedom of exploratory learning, sometimes it led to frustration and inefficient working practices.

Thirdly, part of the process of 'learning by doing' was led by the interaction between pupils and the software itself - a 'create by reacting' response (Zorn, 2009; Victor, 2012). Whilst pupils expressed a preference for learning by interacting with the game under construction, as well as reacting to the components in the software, in practice, they did so only to a limited extent. Some pupils did not actively explore the software or appear to learn much from it. Neither did this 'create by reacting' process bring them into contact with explicit programming concepts which may be required in a formal introduction to programming. They needed teacher intervention in addition. This finding adds to our understanding of computing education in that, although Zorn and Victor champion the affordances of software in the learning process, the current study notes

that for some pupils this ‘create by reacting’ response was limited and suggests that pupils need more guidance to use the software in this way.

10.2.2 Technology focused tasks – big idea no. 2

Constructionist theory promotes the use of technology to generate meaningful products which have cultural value to pupils and asserts that more effective learning is likely to occur when this is the case (Kafai and Resnick, 1996b). The key idea is that pupils should be producers as well as consumers of digital media. Whilst much of the literature cited in Chapter 2 supports this notion, in practice there were some problems, although less is written about the difficulties which arise in such projects.

The presence of the computer and the enthusiasm for hands-on learning meant that some pupils did not make use of the range of resources available to them. Accordingly, they did not learn some useful game design and programming concepts which would have made their games more successful or would have enabled them to avoid problems or correct errors more easily. Whilst pupils found the freedom of the constructionist approach motivating, it was not an efficient way to learn how to use the software for all pupils because some did not make use of the supporting resources available or apply what they had learned in the video tutorials to create their own games. This finding supports related work (e.g. Robertson and Howells, 2008), which recommends a balance between giving children time to explore and work independently with the software on their own terms, and direct, interactive teaching to ensure that essential skills and features are introduced in a timely fashion.

Although some preferred using the software itself to learn how to make a game, the version used in the research (*Game Maker 7*) lacked tutorials within the software and provided little contextual support. Moreover, its graphical interface does not foreground the learning of basic programming concepts explicitly. Pupils learned about ‘objects’, ‘events’ and ‘actions’, encountered programming terms or concepts within the software (variable, parent, persistent) but there was little explanation of how these relate to programming, either in the software or in the teaching resources available for it, and this suggests a need to supplement exploratory approaches with guided interventions.

10.2.3 Hard fun – big idea no. 3

Whilst pupils agreed that the game making project was fun, they also acknowledged that it was more difficult than previous units of work. Yet in the face of problems they displayed perseverance and determination; if they could not solve their problems

outright, they learned to modify or simplify their ideas. They felt a real sense of achievement because they recognised that they had found some success with a difficult task. This validates Papert's claim that pupils enjoy work that challenges them.

10.2.4 Learning to learn – big idea no. 4

The game authoring activity gave pupils some opportunity to 'learn how to learn', a core feature of constructionist learning theory (Papert, 1999a). Making a game showed them that learning in this context is a process of 'cultivation' (Papert, 1993: 104), which takes place over time, where errors are inevitable and important sources for learning, where 'tinkering', making refinements and debugging are common practices. It also showed them that they can learn in different ways (from each other, by doing, by trial and error, by reusing others' code and ideas, by using a range of resources and teaching material, by using the software itself, by researching other games). Whilst similar findings are shared by other studies (e.g. Baytak, 2009), the current research extends those findings to the secondary phase.

In the interview data reported in Chapter 5 pupils expressed a preference for materials and working practices which gave them some freedom and control over their learning and which did not rely on teacher input, because when they were able to work independently it gave them a greater sense of self-efficacy (see section 5.3.2). They valued the fact that they had been given choice about what type of game they made, although this choice created practical problems in terms of resource provision and the level and breadth of teacher expertise required. They enjoyed working in pairs, because they could solve problems together and learn from each other. Having a partner provided a source of feedback and reduced their reliance on the teacher, which they valued.

However, although they enjoyed directing their own working practices and being independent, they did not always balance the requirements of the project - some pupils spent more time on the aesthetics of their game than programming it and this has implications for the pedagogy of game authoring, if it is to be used as a context for learning about programming concepts. This finding is supported by other studies cited in Chapter 2 (e.g. Baytak and Land, 2011b).

A key finding of the research is that pupils prefer 'just in time' support - that is, resources which they can access at the point of need, as they create their games.

Pupils' preference for 'just in time' learning (Riel, 1998) is consonant with constructionist philosophy, which asserts that children do best by finding for themselves the specific knowledge they need - indeed "the kind of knowledge children most need is the knowledge that will help them get more knowledge" (Papert, 1993: 139). But for the children in this study, that was not a straightforward process. Although they liked being given the freedom to choose their own project tasks and expressed a preference for being able to work independently, in order to do that effectively they needed to i) know how to and ii) *be prepared to* find the particular information they needed for their individual purposes. In fact, most pupils did not make good use of the resources made available and were particularly reluctant to use printed learning material. Although they were more positive about the video tutorials, there was a feeling that these were limited in the support they gave and there was a frustration with being 'tied' by these. Pupils preferred to use the teacher as a troubleshooter, as and when needed, and their peers as other forms of 'just in time' support.

When pupils have higher levels of control over their learning and are working independently, at their own pace and from different starting points, and where they are pursuing open-ended tasks, as was the case in the current research, linear patterns of delivery and whole class teaching are less effective and make access to 'just in time learning' resources important. However, providing access to quality, comprehensive, 'just in time' resources aimed at secondary level is a challenge, and providing individualised teaching material significantly so (Fiege, 2011; Herrig, 2013). In the current study, it was difficult for pupils to locate 'just in time' learning resources matched to their level of understanding and preferred formats. An ideal resource would be organised around the core game functionalities required by novice game makers of this age and the programming constructs required to achieve these, and provide visual solutions to and explanations of frequently occurring generic problems or common misunderstandings. Additionally, in open-ended projects of this sort, pupils need to be encouraged to find solutions to problems independently - by accessing a range of physical and online resources, and making greater use of forums or collaborative tools and social media for exchanging their own solutions within the classroom. Whilst all pupils worked independently to some degree, some relied on support from partners, peers and the teacher and did not seek out solutions from other sources, particularly eschewing textual materials. Developing more fully-featured collaborative learning environments, which fuse physical and online resources in a 'sandbox' arena (see for example Frydenberg, 2013) may offer a more enabling approach.

10.2.5 Taking time – big idea no. 5

Constructionism asserts the importance of giving pupils time to learn (Papert, 1991b). While the pupils in this research valued being able to work on an extended project, most games were not completed in the 16 hours available. This is an important consideration for the new Computing curriculum (DfE, 2013c), which covers digital literacy, information technology and computer science (including programming). Extended activities such as game making need to be tightly focused if they are to fit alongside new curriculum content and within the 36 hours per year allocated to the subject at Key Stage 3. In this respect it is important to consider what can realistically be achieved, since it is difficult to complete a fully featured game in the time available, if all stages in the design process are followed.

The extended time allocation was an important feature of the design of the activity, since on a practical level it takes time to learn how to program well enough to create a complex digital artefact, but also because, following constructionist theory, it gave pupils the "time to think, …to get a new idea and try it and drop it or persist, time to talk, to see other people's work and their reaction to yours" (Papert, 1991b: 4). The extended time scale enabled pupils to develop commitment and persistence for learning, and this is identified in the literature as a significant positive outcome of similar projects (see for example Kafai, 1996; Robertson and Howells, 2008; Harel Caperton, 2010). It also gave them experience in dealing with time constraints and modifying expectations, an important part of learning to learn (Kafai, 1996). Yet in terms of the unfinished outcomes achieved by the pupils in this study, the constructionist model was not efficient - and this finding gives support to previous studies which observe that the emergent practices encouraged by constructionist approaches may be ineffective and projects may have a lower probability of success than with more traditional didactic approaches (Hay and Barab, 2001; Ackermann, 2004; Kirschner et al., 2006). It may be that constructionist routes are unsuitable for current time allocations at Key Stage 3.

As a consequence of this, the current research identifies the need to give pupils time to learn how to use and understand the capabilities of the software before they create their own game and the importance of taking time to plan the game interactions before creating the game itself. But in these areas there needs to be a balance between exploratory approaches and more guided teacher support. Pupils need to be given regular project milestones to help them manage their work over time; task requirements

need to be made very clear and aspects of game authoring which are not core to the learning focus of the scheme of work need to be time limited - so for example, graphics and game narratives need either to be provided ready-made, or much more strongly scaffolded, so that more time is made available for the programming of the game and the understanding of programming concepts. The need to more tightly structure game authoring tasks is also identified in the literature (e.g. DEECD, 2010; Doran et al., 2012; Smith and Sullivan, 2012).

10.2.6 Freedom to get things wrong – big idea no. 6

Whilst constructionism asserts the importance of giving children the freedom to get things wrong (as part of the learning process), they need to learn to analyse their thinking, and develop strategies for 'debugging' the errors they make (Papert, 1999a). The findings of this study support that position. Pupils were not adept at analysing their thinking or the problems which they encountered; neither did most of them take a systematic or strategic approach to their work. Some did not view errors as "a source of information" (Papert, 1993: 184) and needed teacher guidance to develop more strategic approaches to identifying and solving them rather than relying on 'trial and error', and this finding is echoed in the literature relating to programming education reviewed in section 2.9 (e.g. Perkins et al., 1986). Indeed, systematic approaches, such as planning, learning to 'read' code and identifying errors, were initially resisted by pupils.

Nevertheless, the game authoring activity gave pupils some awareness of the need for systematic thinking and a real context in which to exercise it. They came to realise the need to take a more strategic approach to their work when they acknowledged that they had not made good use of their plans, that they did not have a clear idea of what they wanted their game to be like, that they had not managed to solve many of their problems. These issues stem from poor program design and lack of checking their code, areas identified in the literature (see section 2.9) as major sources of difficulty for novice programmers which need to be the focus of programming instruction, as much as the learning of language features (Robins et al., 2003).

10.2.7 Teacher as co-learner – big idea no. 7

Papert's vision for constructionist learning environments incorporated the idea that teachers should be co-learners, role models of what it is to be a good learner, showing pupils that it takes time to learn, that errors are part of the process, that learning is sometimes difficult and that we can use a range of strategies to help us. This pragmatic

approach was necessary in the 1980s when the use of computers in education was in its infancy and introducing programming activities was a new venture for many teachers. It is also relevant to the situation ICT teachers find themselves in today as they prepare to deliver the new Computing curriculum.

Yet while the idea of the teacher as co-learner may be compelling and is to some degree inevitable when new subjects are introduced, in practice it does impact on what is learned. In terms of the current research, whilst the teacher's knowledge of computer game authoring was more developed than the pupils' (intermediate as opposed to beginner), it is certainly the case that had it been more advanced still, she would have been able to troubleshoot more effectively.

In other respects, pupils in this study enjoyed the fact that they had more control over their own learning and could work with others. Chapter 5 recorded that some pupils preferred working practices which did not rely on teacher input and favoured learning by doing and being guided by their work as it proceeded. In other words, they enjoyed the constructionist vision of a more distributed and negotiated instruction (Kafai, 2006a) which allowed them to work collaboratively to solve authentic problems.

10.2.8 Using computers to learn – big idea no. 8

Constructionism asserts the importance of pupils learning to use computers and using computers to learn. This was achieved in the sense that pupils i) used software and ii) created software. One of the factors contributing to their enjoyment of the unit of work was the interaction with the software itself. They enjoyed using *Game Maker* because they were using new software as an expressive medium and perceived that it enabled them to be creative, which they valued. The novelty of the software may also have contributed to their enjoyment, in that it developed their sense of competence when they were able to make use of their prior knowledge to help them manipulate new software.

In using a range of software pupils became more practised at using multiple programs simultaneously. They also became accustomed to alternating between areas of *Game Maker*'s integrated development environment (the programming view, the game view, the image editor). Using an IDE was a new experience and broadened their perceptions of what software is and how it functions. As reported in Chapter 2, in creating games with new software tools, young people learn to view games as designed systems and become more systems literate - an important 21st century skill

(Zimmerman, 2009). They also develop practical skills involved in digital media production (graphics, programming) and are thus more able to participate in and contribute to media culture (Pelletier, 2005). Such activities stand in marked contrast to those promoted in the previous programme of study (DfES, 2002a) and go some way to dispelling criticisms of the erstwhile ICT curriculum and its reported over-reliance on the use of office productivity software (see Livingstone and Hope, 2011; Furber, 2012).

Another aspect which contributed to their enjoyment of the activity was that they developed 'construction-oriented' relations with technology (Zorn, 2009; Schelhowe, 2010). They became creators, not just consumers of software:

> CB: When I play on games in the future, I'll think, 'I did something like that'.
>
> SA: I didn't even know you *could* make a game. I've never had any experience of that, ever.
>
> LW: When you play games now you see like …
> JC: You see it in a different way.
> JD: You think about how it's created and that.

This new relationship with technology arises out of the participatory culture and the democratisation of creativity which new technologies have enabled and illustrates the importance of giving young people the opportunity to be producers of digital media, an argument widely supported in the literature (Kafai, 1995; Robertson and Good, 2004; Habgood, 2006; Buckingham and Burn, 2007a; Salen, 2007; Hague and Williamson, 2009; Harel Caperton, 2010; Li, 2010).

Finally, some pupils valued the game authoring activity in terms of its relevance to the games industry and saw value in the project as a preparation for possible future careers. This finding is important in light of concerns that the needs of industry were not well served by the previous ICT curriculum, which, according to some commentators, focused solely on basic digital literacy and office skills (see Livingstone and Hope, 2011; Furber, 2012).

This section has discussed the game making process in terms of the eight big ideas of constructionism, which set out a rationale for what Papert considered to be the 'proper' use of computers in education. That is, computers should be used as a means to

access powerful ideas and complete challenging projects, pertinent to the digital world we live in, and to create personally meaningful artefacts. In the course of this purposeful, practical, computer-based activity pupils encounter new ways of learning and working, which arise out of being given time to appropriate their work (cognitively and affectively), having the freedom to explore ideas in their own preferred styles and recognising that getting things wrong is part of the learning process. This new way of working generates new ways of thinking and new relationships with technology. This section has illustrated how pupils' perceptions of the process they followed largely validate the constructionist position.

10.3 Making games – the outcomes

Chapter 6 addressed the research question 'What are pupils' perceptions of the outcomes of the game authoring activity?' It records that pupils found game making a motivating context for learning in a range of areas although they grappled with the challenges of creating satisfying outcomes in terms of graphics and playability and experienced difficulties with creating a convincing game narrative.

10.3.1 Difficulties with game design

The scheme of work followed a generic systems development life cycle, and as part of the design stage, pupils created a design document, a storyboard, and planned the game interactions. Some pupils expressed impatience with these tasks and did not fully complete them because they wanted to begin making their games straight away in the software. This made the subsequent implementation of their games more difficult. The findings of this study suggest that conceptualisation is important to successful implementation, since those who planned their games more thoroughly at the start, in terms of the storyline, visual appearance and game interactions, made better games, in so far as the graphics were more effectively presented and the gameplay was more functional and coherent. Although constructionism promotes epistemological pluralism (Turkle and Papert, 1990), in which 'bottom up' approaches are valued, the findings of the current study suggest that without planning, both game design and programming are compromised.

The analysis of the planning documents identified several areas where pupils needed further support. For example, pupils were asked to outline the visual appearance of their game in a storyboard, yet significant omissions in graphical information were made in these. Player characters and other game objects were not always clearly

represented and the storyboards did not give a strong visual sense of what the games should look like.

Significant details, such as the use of sound and game controls were also omitted in some of the design documents; game play, level progression and win/lose states were not described by all. These omissions indicate that pupils found it difficult to visualise these aspects in the detail required, because they were not accustomed to designing digital media, or because computer-based activities were preferred to paper-based planning tasks. Future implementations would more strongly model planning and design activities, and intersperse these with practical work, so that a stronger connection is made between the design documents and the games themselves.

Those pupils who did not fully complete the planning documents and those who had departed from them did not have a clear idea of what they wanted to achieve and found their games more difficult to implement. In terms of constructionist learning theory this illustrates that 'bottom up' modes of working are not always enabling; while Papert suggests that those who are guided by their work as it proceeds can do as well as those who follow a pre-established plan (Papert, 1991b: 6), the findings of this study do not support this view. Whilst some pupils preferred learning by doing, and resisted planning, this approach was not entirely successful in terms of the outcomes they produced.

These findings offer insights into the areas of game design which pupils found difficult to manage and so inform future implementations of game authoring curricula.

10.3.2 Narrative

Although pupils valued being able to choose the genre and storyline for their own game, in practice this created difficulties. Their initial ideas had to be simplified since they did not have the curriculum time or the level of programming skill required to complete the sophisticated games they at first conceived. Initial ideas were not always clearly articulated or adhered to and this created problems at the implementation stage. Pupils who did not develop a clear narrative for their games found it more difficult to visualise and implement the game play and interactions.

Pupils had difficulty with creating a coherent, believable narrative, which was suitable for the target audience and found generating specific details, such as the rewards, obstacles or enemies for their game, or how the game should progress from one level

to the next, challenging. It seems that the construction of a game narrative was a complex task for pupils and added an unexpected layer of difficulty to the activity, a finding not reported elsewhere. This contrasts with studies cited in Chapter 2 which suggest that making computer games develops pupils' narrative abilities (see Robertson and Good, 2004, 2005, 2006; Burn, 2007).

Since the focus of game authoring in the ICT curriculum is not on narrative development, the findings of this research suggest that for future implementations of game authoring schemes of work, pupils should develop prototype games where the emphasis is on developing game mechanics and object interactions (and the programming required to achieve that), rather than on the storyline or graphics. Other solutions are to use generic game formats, such as *Break Out* or *Pacman* or to provide pupils with partially complete, ready-made games or templates which they can then modify and personalise. Alternatively, to reduce the need to devise a narrative from scratch, game narrative outlines could be given, based on a social/environmental issue, a model promoted by 'serious games' organisations (Apps for Good, 2013; Games for Change, 2013), and online game-making courses (e.g. Macklin, 2010), or curriculum subjects could be used to frame pupils' game ideas, which is the context given for creating games in several studies cited in Chapter 2 (for example, Harel, 1991; Kafai, 1995; Baytak and Land, 2011b) and in current examination specifications (OCR, 2009a; AQA, 2012b).

10.3.3 Graphics

One of the main outcomes of the game authoring activity was that pupils learned to create and edit graphics using *Fireworks* (Macromedia, 2004) and developed skills and understanding in concepts important in this area. However, findings suggest that although it may be important to give pupils an opportunity to create their own game graphics because they enjoyed doing so, in practice it was a time-consuming task. Pupils used a mix of graphics sourced from the internet and those they created themselves, with varying levels of success (see Appendix 1). Pupils acknowledged that they spent too much time on the graphics, which meant that there was less time to program the game action, a finding also widely reported in the literature (see Kafai et al., 1997; Shackleton et al., 1997; Lin et al., 2005; Parsons and Haden, 2007; Willett, 2007; Northcott and Miliszewska, 2008; Baytak et al., 2011; Macklin and Sharp, 2012).

The professional graphics software used had a steep learning curve, and an unfamiliar interface, so even basic tasks such as resizing graphics or creating transparent

backgrounds were new areas of learning for pupils. This has implications for schemes of work. Since the creation of graphics is an important part of creating digital media, this area of learning needs to be incorporated into the new Computing curriculum, so that skills are systematically developed across the Key Stage. This finding gives support to related research into the practicalities of digital media production (see Willett, 2007), which similarly notes that graphics software requires significant formal instruction in order to become a creative tool for young people. At the very least, pupils need more practise in using basic drawing tools and developing image editing skills in each of Years 7, 8 and 9, supplemented by targeted lessons on how to create the sorts of graphics used in two dimensional computer games and other screen-based media - titles, interaction buttons, backgrounds, and sprites (static and animated). In the absence of this, schemes of work should direct pupils to make use of *Game Maker's* image editing tool to create 'pixel art' graphics or use ready-made sprites, or 'placeholder' sprites, to reduce the complexity and duration of graphics tasks.

10.3.4 Usability

Authoring a game gave pupils an authentic reason to design for usability. Usability in the previous Key Stage 3 ICT programme of study (QCA, 2007b) referred to meeting the needs of an audience and ensuring fitness for purpose. Usability in the context of creating a computer game was extended to notions of functionality and player experience. Pupils showed an awareness of the need to use conventional game controls and to provide user options, such as game exit/restart. They learned about other aspects of usability, such as the importance of title screens, game instructions, and game navigation to enhance player experience. These aspects of usability are identified in the literature as important areas of design which need to be considered when evaluating the learning opportunities afforded by computer game authoring and when analysing the games pupils make (see Reynolds et al., 2010; Denner et al., 2012; Wilson et al., 2012).

10.3.5 Interactivity

Envisioning and creating interactivity was a new area of learning for pupils and it was this distinctive quality of computer games which set the activity apart from other ICT projects. Previous experience of designing interactivity consisted largely of creating clickable buttons (e.g. for web site navigation). In the game-making activity pupils' involvement with interactivity became deeper since they had to think about how to transpose their game narratives into dynamic representations and deal with the complexity of creating multiple interactions between game objects. Pupils found this

challenging. Its difficulty partly arises because creating interactivity usually requires some form of programming. The following section summarises the programming concepts which pupils encountered as they endeavoured to build the interactivity needed to make a playable game.

10.4 Learning to program

Chapter 8 addressed the research question 'What difficulties do pupils have with game programming?' and records that although *Game Maker* was designed to enable users to create computer games without the need to learn a 'difficult' textual programming language, the children in this study still found some aspects challenging. Programming errors most frequently occurred due to a lack of precise, logical thinking and a lack of testing/checking. Pupils were not used to thinking algorithmically, decomposing problems, or reading and evaluating their code.

10.4.1 Program design

Before they began to make their games, pupils were asked to plan the game interactions by listing objects and specifying the events and actions assigned to them. Some pupils did not complete this task effectively because they were unaccustomed to decomposing programs into their constituent parts, and were not practised in applying precise, logical, systematic thinking when planning the interactions in their game. They were also reluctant to spend time on planning tasks because they wanted to begin making their games. Yet the findings of this study indicate that without planning, both game design and programming are compromised. This finding is supported in the literature, which identifies program design and planning as a key component of learning to program (see section 2.9).

Their initial plans were characterised by incompleteness (not all objects in the game were listed, not all events or resulting actions were visualised or described). Pupils sometimes conflated events and actions, did not break down object behaviour into separate events, or assigned multiple actions to one event, instead of to separate, distinct events. This 'merging' of separate processes is found to be a common source of error in novice programmers (see Spohrer and Soloway, 1989) according to the literature (see section 2.9).

Later in the planning process, pupils began to separate events and actions, and introduced a wider range of inputs into their plans (for example, they included non-user inputs such as conditional statements, as well as user-controlled inputs, such as a key

press). This suggests that they were beginning to 'think computationally', and that an understanding of the need for decomposition and precision in program design was beginning to emerge.

However, most pupils appear to need support with being specific and precise at the planning stage, in listing the objects, events and actions themselves and in using the correct terminology to refer to them. This study suggests that more emphasis needs to be placed on program design, so that pupils effectively plan the game interactions, before they build their game. This aspect of learning does not receive much attention in the studies cited in the literature review (but see Doran et al., 2012), although some research acknowledges children's reluctance to engage in or make use of planning work and their preference for focusing on aspects which give immediate feedback and satisfaction, such as graphics and animation (see Howland et al., 2013).

10.4.2 Learning programming concepts

Chapter 7 addressed the research question 'How does game authoring using *Game Maker* support the learning of basic programming concepts and practices?' and showed that using the software introduced pupils to several basic programming concepts and gave them an understanding of the precision and detail required in constructing game programs.

Chapter 7 records that *Game Maker* gives good support for the understanding of event-driven programming, since users have to select events to initiate object inter(actions). Whilst pupils were used to the idea that the keyboard and mouse are input devices, in making a game program they learned that inputs can be controlled by non-user events, such as collisions, conditions and other game states. Some problems occurred with the use of events because some pupils did not understand the domain specific meaning of 'event' and sometimes confused *events* with *actions*, chose the wrong event, duplicated events in more than one object, or used conflicting events.

Using *Game Maker* also introduced pupils to the concept of object-oriented programming - they learned that a game is made up of objects, which are programmed entities. However, while they found it straightforward to view the player character and other game resources as objects, some found it more difficult to understand that interface controls, such as 'start' buttons, were also programmable objects. Some pupils did not initially understand that all game components are separate entities and that game objects should not be drawn as part of the background or that the visual

appearance of the game (sprites, backgrounds) is separate from its underlying functionality.

All pupils used actions in their games and in so doing gained practice in sequencing instructions. Using actions also taught pupils that mathematical concepts (e.g. coordinates, negative and positive number, angles, probability) are important for game programming, since they are often required to set parameters and arguments for them. Errors sometimes occurred when pupils duplicated or used conflicting or incomplete actions, or had difficulty in setting the correct parameters or arguments to achieve required behaviours.

Pupils also learned about the programming concepts of *selection* and *iteration.* Six pairs used conditional statements and some form of 'loop' construct successfully in their games. However, others found these constructs difficult to implement, suggesting that they need to be formally taught and that aspects of games which make use of these constructs need to be clearly modelled if they are to be successfully used by all.

All pupils used variables (for example, 'score' and 'speed') but since many of these are inbuilt and therefore 'hidden', pupils may have used them without understanding. Most pupils did not use the term 'variable' to refer to these features and only one pair created a variable after following a tutorial. These findings suggest that the concept of variables and the role they play in computer games needs to be explicitly taught when using *Game Maker*.

The findings in Chapter 7 suggest that using *Game Maker's* drag and drop environment to introduce basic programming concepts such as conditionals, loops and variables, is only partially successful. Pupils are unlikely to learn these concepts without direct instruction or modelling and without the appropriate programming terms being emphasised. Project briefs need to specify key programming concepts required for a game. For example: a score must be set to introduce the use of variables; a score must be tested to illustrate the use of a conditional statement; an action must be repeated to show the application of a loop in a game program, and so on. These constructs need to be clearly demonstrated in *Game Maker* before pupils can implement them.

The need for direct instruction is significant. The theory of constructionism, which underpins this study, suggests that 'learning by doing', constructing one's own understanding by constructing a computational artefact, and exploratory learning are

valid ways of working. However the findings in this study suggest that such approaches may not be appropriate for learning programming concepts and this idea is supported in several studies cited in Chapter 2 which also suggest that some programming concepts need to be formally introduced if they are to be used effectively (see Kelleher and Pausch, 2007; Maloney et al., 2008; Kuruvada et al., 2010b; Schelhowe, 2010; Denner et al., 2012). The findings of the current research also support research which makes a similar claim for other programming environments (see Ben-Ari, 2001; Beynon and Roe, 2004; Beynon and Harfield, 2010; Meerbaum-Salant et al., 2011), and which suggests that constructionist approaches are inefficient when it comes to learning about programming concepts and are not well suited to the early stages of learning to program (Guzdial, 2009). While some studies support the idea that bricolage is a valid way to learn programming concepts *for some learners* (McDougall and Boyle, 2004; Stiller, 2009), the findings of the current research suggest that this is not likely to be an effective way to maximize the learning of core programming concepts for the majority of pupils. This contrasts with the findings of earlier research of children making games and learning programming in *Logo* (see Harel, 1991; Kafai, 1995).

10.4.3 The language of programming

In their initial planning documents, most pupils did not appropriate the language of *Game Maker*, or the terms they had come across in the video tutorials, which made their plans less supportive to them later in the implementation phase. Some pupils misinterpreted the context specific meaning of words like ‘event’, ‘action’ and ‘room’. For example, they understood the word ‘event’ to mean ‘something which happens’ in the narrative of the game, rather than as an input. This misunderstanding of natural language terms in programming contexts is identified in the literature as a common source of error in novice programmers (du Boulay, 1986; Pea, 1986). However, 4/12 pairs used correct terminology in their plans; these pupils also produced the most complete games.

These findings underline the importance of using correct terminology to refer to programming concepts when using visual languages such as *Game Maker*, especially where those terms are hidden by the software. For example, *Game Maker’s step events* or *alarm events* hide the program iterations/loops which they generate; *test/check* actions hide that they are conditional statements; common variables are score, room width/height, x/y position, speed and these are set by default for all objects - but the word ‘variable’ is not used to refer to them. These terms need to be drawn out by the teacher and emphasised in learning resources.

Even though some programming terms are not made explicit in *Game Maker*, making a computer game introduced pupils to some aspects of the language of game design (collision, sprite, room, challenge, goal), programming (objects, events, variables) and also more abstract words to describe states, behaviours and interactions (solid, visible). Some pupils enjoyed using this domain-specific language and became increasingly fluent in it. New words gave them access to new concepts and pupils began to use these words as their understanding of computational concepts emerged, such as one pair who confidently discussed their use of variables. This exposure to the discourse of game design enhanced their digital, media and systems literacies, identified in the literature as important 21st century skills (see Games, 2008b; Zimmerman, 2009; Harel Caperton, 2010).

However, not all pupils found this 'new language' easy to embrace. For some the specialised language was a barrier and they avoided using actions whose referents they did not understand, for example, and did not make use of error message text or action definition text to further their learning.

This suggests that the 'language of programming' needs to be made more explicit in schemes of work using *Game Maker*. Key words in programming need to be brought into use early on. Pupils should be encouraged to use technical terms in their design documents and throughout. Teachers need to articulate the programming knowledge that pupils have acquired, almost without knowing it (Good, 2011), by drawing attention to the language of *Game Maker's* event selector and action icons, particularly the core programming constructs of conditions, variables and loops. To do so gives pupils an insight into some of the building blocks of computer programs and key areas of game design, and demystifies the language used. As pupils begin to use the vocabulary and language of programming, so they begin to think computationally (Grover, 2011) and realise that use of precise language is important for learning to program (Fletcher and Lu, 2009; National Research Council, 2009).

10.4.4 Code reading/program comprehension

Whilst *Game Maker* provides a concrete, visual representation of programming constructs, the findings of this study suggest that some additional theoretical input is necessary to ensure that pupils have understood the underlying concepts. This can be achieved by encouraging pupils to read the textual information which corresponds to the graphical code they produce and to annotate the programming constructs they use. In so doing, pupils practise using programming terms and interpreting the pseudocode

equivalent to the visual action icons they select. Pupils need also to be encouraged to add comments to their code to articulate their understanding of it. These practices encourage them to develop/check the logic of their games and take them one step closer to expressing code in a textual format. Such recommendations are missing from the literature cited in Chapter 2 surrounding the use of *Game Maker* (e.g. Baytak and Land 2010; Doran et al., 2012), none of which considers the use of *Game Maker's* textual object information or code commenting as part of programming pedagogy.

To support the development of their own games, whilst emphasising programming concepts, schemes of work need to incorporate a range of scaffolded activities - for example, provide code walkthroughs for common game mechanics, similar to *Scratch* cards (Rusk, 2009), introduce code reading/code debugging exercises, and code writing tasks, where pupils work with partially completed programs to correct errors or extend functionality. This would ensure that pupils' preferences for practical work are met at the same time as introducing programming concepts and providing targeted support for making their games. While such approaches have been successfully used in studies related to the use of *Game Maker* cited in Chapter 2, (e.g. Guimaraes and Murray, 2008; Hernandez et al., 2010), none of the currently available tutorials or textbooks aimed at the education market focus on this aspect - they guide pupils to make a game - but do not focus sufficiently on the underlying programming concepts. For example, a recently added 'Learn' section on the *Game Maker* website (YoYo Games, 2014) provides video tutorials for how to make 3 games, but the underlying programming concepts are not drawn out, as is required for the new programme of study for Key Stage 3 Computing, nor are these aimed at a Key Stage 3 audience.

10.4.5 Computational thinking

The findings suggest that in creating a game using *Game Maker*, some pupils began to think computationally, a core feature of the new Computing curriculum (DfE, 2013c; Kemp, 2014) and an important 21st century skill (Wing, 2006; Wing, 2008; Perković et al., 2010; Repenning et al., 2010; Barr et al., 2011; Brennan et al., 2011; Denner and Werner, 2011; Google, 2011; Kane et al., 2012). In particular they became aware of the need to decompose a game into its constituent parts (sprites, objects, sounds, rooms, backgrounds), and to decompose game play into separate sequences.

In using a new programming paradigm, some pupils were able to make links between their prior learning in *Flowol 3* (Bowker, 2005), and their learning in *Game Maker*. They saw that the two programming languages shared common ground, even though the

visual representation of programs was different and they were designed for different purposes (to control simulations of physical systems; to create computer games). Thus, their understanding of what constitutes a computer program was somewhat expanded.

In using *Game Maker*, pupils learned that computers are deterministic - they do only what they are programmed to do. They came to understand this when they discovered that they needed not only to program an object to move, for example, but also to stop it moving, and that they had to set a speed, as well as define a direction, for movement to occur. This gave them some awareness of the level of precision required in writing programs, identified in the literature as one of the areas of difficulty in learning to program (see section 2.9). In a similar vein, pupils were surprised when game objects disappeared from view and did not stop at the edge of the screen. They did not initially understand that the boundaries of the computer screen are not recognised by game objects. At this stage in their learning they had not developed an effective mental model of how the game space is defined by coordinates and that these, not the physical computer screen, set the dimensions of the playable space. Building effective mental models of the programs they create is identified in the literature as an area of difficulty in learning to program (see section 2.9) and an area which, the findings suggest, requires more attention in game authoring schemes of work.

Nevertheless, in creating their games, pupils were introduced to some important computational thinking concepts and gained some awareness of how digital media are constructed. These areas of learning are important in constructionist projects, because using technology as 'building material' brings pupils into contact with ideas important in a digital world, and 'knowing about digital technology is as important as reading and writing' (Papert, 1999a: n. pag.).

10.5 Affective values of authoring computer games

Chapter 9 addressed the research question 'What affective value is there in authoring computer games?' and reported perceived values in the affective domain (motivation, enjoyment, confidence) and in learning and thinking skills, such as perseverance, and independence, which were observed during the game-making activity. Constructionism sees an important role for affect in learning, arguing that pupils are more likely to become intellectually engaged when they are working on personally meaningful activities. This positive relationship with learning is as important as what is learned

(Kafai and Resnick, 1996b: 2), and is more likely to occur when pupils are encouraged to develop and value their own ways of working (Turkle and Papert, 1990: 135). This section discusses perceived values in the game authoring activity, beyond achievement in the cognitive domain.

One of the positive outcomes of the activity was that pupils enjoyed making a computer game. Pupils valued the games they produced because they had created something of contemporary, cultural and social significance, which had personal meaning for them. Importantly, they experienced a different relationship with the outcomes of their learning, because in making a game for others to play they saw a real purpose in their work. This sense of achievement is echoed in the literature cited in Chapter 2 relating to the motivational affordances of game authoring, and pupils' satisfaction with creating an authentic product is a validation of constructionism's defining proposition (see Papert, 1986).

Pupils enjoyed the activity also because notions of 'play' and 'fun' were uppermost - a game is created for others to play and enjoy. This contrasts with the more serious, functional purposes of other systems pupils had previously developed (control systems, booking systems). The word 'fun' was frequently used in the transcript to describe some aspect of the game-making activity.

Pupils enjoyed the activity because the mode of learning was playful. 'Work' became a process of experimenting, creating and playing. If mistakes were made they could be corrected on the fly. In this respect, the 'fun' they refer to is an outcome of the affordances of the software and the collaborative working pattern, features of constructionist learning environments.

Making a computer game was also a new mode of creative expression for them. It was the first time they had created a game and also the first time they had made 'software'. They enjoyed the variety of activities involved (creating the visual appearance of the game, locating sounds, programming objects, developing scoring systems). This creative aspect of game authoring is identified in the literature as one reason why pupils find it enjoyable. The fact that the games created are also playable makes it a unique form of creativity (Buckingham and Burn, 2007b).

Their enjoyment of the activity also led to greater levels of engagement, which was evident in pupil time on task. Most pupils seemed immersed in their work:

> AC: When we are creating a game we come in and sit down and we get on with our [work] straight away.

This finding is validated in the literature, which widely reports that young people find game authoring motivating and that this in turn leads to positive attitudes to learning (see Howland et al., 1997; Chamillard, 2006; Kafai, 2006a; Denner, 2007; Sanford and Madill, 2007b; Repenning and Ioannidou, 2008; Cheng, 2009; Jung and Park, 2009; Li, 2010; Fowler and Cusack, 2011).

Some pupils gained in confidence because their out-of-school interest in computer games was able to find expression in the classroom. They felt that they had some expertise to bring to bear on the learning activity. In this respect game authoring bridges their use of technology out of school with that in school (see Buckingham et al., 2003). Others became 'experts' with *Game Maker* and were consulted by their peers for the first time, and this is likely to have increased their self-esteem. These findings give support to several studies cited in the Chapter 2 which report positive effects in terms of attitudinal improvements to learning and gains in confidence, particularly for those lacking in engagement (for example, see Robertson and Howells, 2008; Baytak and Land, 2010; DEECD, 2010; Li, 2010; Fowler and Cusack, 2011; Passey, 2012).

The positive attitudes relating to game authoring reported in this section are clearly of value, and belie recent characterisations of ICT curricula as offputting, demotivating and dull (see Peyton Jones, 2010; Furber, 2012; Gove, 2012b).

10.6 Implications of the research

By the end of the 16 hour activity, only 3/12 games (AEMD, ACJC, JBLA) were 'finished', in so far as they functioned without significant problems and enabled the player to achieve a score and progress through one or more levels. This has implications for the place of game authoring in the Key Stage 3 ICT/Computing curriculum, typically delivered in one hour a week in each of Years 7-9, and suggests that the model used here, where pupils designed and programmed a game from scratch and created the game graphics themselves, may not be the most suitable approach. Decisions have to be made about the learning focus of game authoring projects (i.e. programming or graphics, but not both together), so that they can be

completed within the time available, and so that not too many competing demands are made on learners.

Leading on from this, there are implications for curriculum planning. Schools may have to embrace alternative models of curriculum delivery if they wish to promote extended design and programming activities such as game authoring. Willett (2007) questions the feasibility of young people creating computer games at all, since to do so demands high levels of skills in a range of areas and extended time scales, yet may produce limited outcomes. Out of school clubs or intensive, week-long enrichment activities may offer more suitable sites for learning, or it may be necessary to dedicate more time to game-making activities, as suggested in online programmes such as *Globaloria* (Harel Caperton et al., 2010) and *Gamestar Mechanic* (E-Line Media, 2013).

The research also has implications for the assessment of such activities. There was wide variation in the quality of the games, and within the games, between the levels achieved in the different components (graphics, sound, programming, game play). Some games had little functionality and were more akin to animations; others had 2 or 3 playable levels. For the purposes of this study, to evaluate the learning evidenced in the games, eight assessment criteria were given a 'score' (aligned with the five levels of the SOLO taxonomy (Biggs and Collis, 1982)) and a total was calculated (see Chapter 4). To indicate the range in achievement and to illustrate that all games evidenced some learning, Figure 36 below charts the overall 'score' attained by each pair.

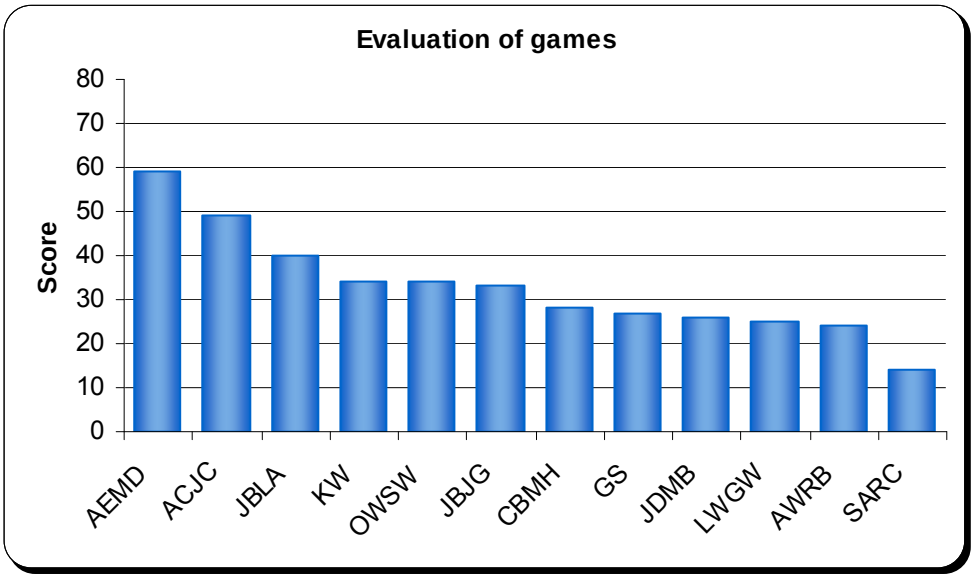


Figure 36: Evaluative score for games

However, the procedure used to evaluate pupils' games in this study would be impossible to replicate in mainstream classroom contexts and this raises the question of how extended programming projects such as these should be assessed.

The wide range in outcomes further suggests that constructionist approaches are not suitable for all learners, especially those who need more guidance and structure. While most pupils in this study had an above average ability profile (see section 4.4.3), they did not all display independent learning behaviours or make use of the sources of support made available to them, and this may account for the variation in the games produced. Those who made better games tended to be more able pupils (6/9 were of high or above average ability), but their success seemed to have as much to do with their willingness to learn independently as to do with their cognitive ability. This variability in pupils' readiness to learn independently may also reflect the extent to which they had or had not encountered similar project-based activities in other areas of the curriculum.

Constructionist approaches may also not be well-suited to some elements of game authoring. Some aspects of learning, such as the development of graphics software skills, or the learning of programming concepts need, at this level, to be formally taught if they are to be successfully used by all - for these areas of learning, learning by doing and experimentation alone appear not to be sufficient. Pupils also need to be guided to complete tasks which are not immediately popular, such as planning the game program and object interactions.

The learning evidenced in the games of course only partially indicates whether or not the game authoring activity was worthwhile. Other positive outcomes are to do with improved attitudes to learning and affective gains and the collaborative working practices which grew out of the activity. These have been discussed in Chapters 5, 6, and 9 and together appear to offer a persuasive argument to include such curricula in mainstream UK ICT/Computing settings, despite the challenges of doing so.

In spite of these implications, this study has shown that in making a game pupils learned to use some basic programming constructs and began to think computationally, although this could be further developed by:

i) Spending less time on graphics at the expense of developing the game program.

ii) Putting more emphasis on the language and practices of programming (use of the discourse of programming, program design, precise logical thinking, systematic program checking/testing).

iii) Enabling pupils' preferred way of working 'learning by doing' by providing age-appropriate, 'just in time' resources in a range for formats which they can access at the point of need to help them solve their individual problems and to support understanding of the underlying programming concepts.

iv) Encouraging greater use of online collaboration and communication tools.

These recommendations are based on lessons learned from the study, drawn from the data in Chapters 5-8. The next section builds on these to consider the broader contributions of the research.

10.7 Contributions of the research

i) This study has explored whether constructionism is a suitable approach for learning how to make a computer game, involving the domains of design and visual programming. The findings suggest that as an approach to learning, constructionism appears to have yielded positive effects in terms of affect; the collaborative learning environment which developed in the classroom and the high levels of motivation and engagement reported by pupils are positive outcomes of the activity and give support to the constructionist learning theory which frames it. But the findings also suggest that more structured interventions are needed with regard to learning basic programming concepts, and core game mechanics (see section 10.4 above) to ensure that key game functionalities and underlying programming constructs are demonstrated and understood.

The findings give support to previous research using different programming environments, which suggests that constructionist approaches are not well aligned to learning programming (Beynon and Harfield, 2010) since the syntax and semantics of programming languages are non-negotiable (Beynon, 2009: 73). Bricolage is also criticised because it leads to "endless debugging" and is therefore neither an effective methodology nor an effective epistemology for programming unless it is supplemented with planning and formal methods (Ben-Ari, 2001: 66). Other studies express cautious support for constructionist approaches by suggesting that bricolage is a valid way to

learn programming concepts *for some learners* (McDougall and Boyle, 2004; Stiller, 2009), and the findings of the current study bear this out.

ii) This research makes an original contribution to the field of computer science education, since little research in this area has been targeted at secondary level (Begel and Klopfer, 2004). Moreover, there are few studies which look at the learning of computing concepts through game authoring *within a classroom setting* (Wilson et al., 2012) or what kind of knowledge students have learned from creating games using visual programming languages (Koh et al., 2010).

Chapter 2 notes that *Game Maker* is widely used at secondary level in UK schools, as evidenced in the textbooks and examination specifications which refer to it. However, few studies focus on how *Game Maker* is used to teach game authoring in the UK secondary ICT curriculum (Hayes and Games, 2008; Daly, 2009) or what may have been achieved in terms of learning basic programming concepts and computational thinking for pupils in Key Stage 3 (Denner et al., 2012). This study adds to the knowledge base surrounding the use of *Game Maker* in the secondary UK IT/Computing curriculum in these respects. Its unique contribution is that it presents a detailed account of the programming concepts which pupils encounter and the difficulties they have with these. In particular it highlights the need to use the language of programming right from the start and to place more emphasis on program design and planning.

Few recent studies focus on the errors pupils make with visual programming tools. While Doran et al. (2012) suggest promoting a strategic approach to error handling by including 'guided errors' in their programme, they do not identify the errors pupils actually make. The current study usefully extends the findings of studies cited in Chapter 2 which identify the programming constructs pupils use or do not use (e.g. Denner et al., 2012), and the areas of difficulty they have with game programming using different software (e.g. Good et al., 2010).

The analysis of the planning documents identified several areas where pupils needed further support and this adds to our understanding of game authoring pedagogy; other studies in game authoring do not investigate the elements which pupils either include or omit from their design documents.

iii) In focusing on the use of *Game Maker*, this study adds to the body of research which considers the impacts of particular tools on the learning of basic programming (for example, Pea, 1983; Pea and Kurland, 1984; Kurland et al., 1987; Mendelsohn et al., 1990; Meerbaum-Salant et al., 2011; Stolee and Fristoe, 2011; Adams and Webster, 2012; Werner et al., 2012a).

iv) The research also pays attention to a growing area of interest in the current UK secondary ICT/Computing curriculum - the pedagogy of computer game authoring and programming. It describes the introduction of a particular unit of work in game authoring, which provides an interesting picture of pupils' perceptions of the activity, the areas of learning they encountered and the difficulties they experienced. As a result of these insights the study makes practical suggestions for how to improve the delivery of units of work which use *Game Maker* to teach basic programming concepts and practices at Key Stage 3. It therefore extends the knowledge field in studies of game making, which are predominantly situated in the primary phase (see Chapter 2) and makes a useful contribution to the pedagogic content knowledge (Mishra and Koehler, 2006) of game making and programming, which is currently under researched (Saeli et al., 2011). In particular, this research focuses on making games as part of the ICT/Computing curriculum, and this extends the reach of much of the literature cited in Chapter 2, which is concerned with authoring computer games to enhance learning in other subjects.

v) Whilst much of the literature cited in Chapter 2 supports the notion of pupils becoming producers of digital media, less attention is paid to the difficulties which arise in such projects. This study identifies some problems which arose in practice and makes suggestions for how to avoid these.

vi) The scheme of work followed in the research and the findings which arise out of its implementation make a useful contribution to current debates about the pedagogy of programming, especially taking into account the training needs of teachers of ICT who do not have a computing background, but who now have to teach Computing (DfE, 2013c). This research illustrates how introducing basic programming concepts using *Game Maker* may be a viable approach for teachers and pupils who have little prior knowledge of the field and makes recommendations for how to bring those concepts to the fore to achieve a balance between the aesthetic and functional aspects of computer game authoring.

vii) The research also generates a framework for the assessment of computer games made by pupils and targeted at the Key Stage 3 level, based on the SOLO taxonomy (Biggs and Collis, 1982). Table 3 presents researcher-developed criteria for the assessment of computer games made in *Game Maker,* incorporating the domains of game design and programming. Such alternative approaches to assessment are particularly useful at the current time given that the attainment target for ICT has been disapplied and schools are expected to select their own assessment methods (DfE, 2013a).

viii) Importantly, the research focuses on mainstream school settings, in contrast to much of the literature cited in Chapter 2 which is situated in out-of-school contexts. As a corollary of this, it considers the possibilities for creating games within limited timescales within the 'everyday' curriculum, in contrast to the work of others which spans much longer intervals or is conducted as intensive research projects (see Harel, 1991; Kafai, 1995; Harel Caperton et al., 2010).

ix) Finally, the research provides a methodology for analysing computer games *as data* (see Chapter 4), an area which is not widely covered in research methods literature. The methodology for analysing the learning in programming evidenced in the games involved using *Game Maker's* object information to determine which aspects of code were correct or incorrect, alongside focused game play sessions to record functionality and playability. The analysis developed rubrics for the evaluation of computer games as i) designed and ii) programmed artefacts (see Table 1 and Table 2) and developed criteria for the assessment of computer games incorporating these two domains (see Table 3).

10.8 Limits of the research

Despite the contributions made by the research, it also has its limitations:

- The research was conducted with one pilot group (n=23) and one main study group (n=22) in a high-achieving school in an affluent area of South East England. Its findings may not be replicable in different settings.

- Although the group was mixed ability, 10/22 pupils achieved 'above average' values in their average CAT scores; 7 pupils achieved a CAT score of 120 or

higher in one or more CAT measures, which suggests that the group was of above average ability. Its findings may not be replicable in different populations.

- The study represents one implementation of a scheme of work for computer game authoring, using *Game Maker*. It is acknowledged that the particular scheme of work, the game authoring software, and resources made available to the pupils in this study will have delimited their learning of programming concepts. Its findings may not be replicable using other software.

- The small scale of the study limits the reliability and the validity of the findings in so far as additional findings may emerge in larger populations. Its findings are best evaluated as *one amongst other* case studies of game authoring projects which investigate different tools and settings (see for example, Kafai, 1996; Lavonen et al., 2003; Willett, 2007; Robertson and Howells, 2008; Zorn, 2008; Games, 2010; Hernandez et al., 2010; Baytak and Land, 2011b; Kafai and Peppler, 2012; Macklin and Sharp, 2012; Minnigerode and Reynolds, 2013).

While these are limitations they do not negate the insights into the pedagogy of computer game authoring gained by conducting this research (see section 10.7). The local, small-scale, particular features of the present study hold value, since "phenomena are … present in the smallest particulars of practices and institutions" (Maclure, 2006: 230) and can make a useful contribution to the field, or prompt further research of a larger scale.

10.9 Future work

The findings of the research set the groundwork for further investigation in the following areas:

1. The development of a framework for computer game authoring for Key Stage 3 which foregrounds the learning of programming concepts using *Game Maker*. The development of age-appropriate physical and online resources to support this, matched to pupils' preferred formats and focused on: how to implement key functionalities of games; clear explication of programming constructs used to achieve these; access to sample code in visual format and greater use of online tools to support collaboration and peer-learning. As an extension to this, the development of a unit of work which

shows pupils how to make a game in *Game Maker's* textual programming language, GML. Currently available resources do not take this approach and provide limited support for the theoretical understanding of programming constructs (for example, see Yoyo Games, 2014). Further research would evaluate such pedagogy.

2. Research into how to assess pupils' understanding of the programs they create when authoring games, and their achievement in other aspects of game design is a fruitful topic for further investigation, particularly in light of the disapplication of the attainment target in 2013 (DfE, 2013a).

For the purposes of analysing and evaluating the games pupils made in the present study, a modified version of the SOLO taxonomy (Biggs and Collis, 1982) was devised, to incorporate elements of programming, game design and functionality (see Table 3). Detailed analyses of the games yielded quantitative data about programming constructs used (see Table 2) and descriptive accounts of game design attributes evaluated elements of game design (see Table 1). Pupils' SOLO levels for each component were mapped to give an overall picture of the level at which they responded to each component. However, such a detailed assessment would be very time-intensive for teachers in mainstream settings.

Artefact-based interviews were recorded with 7 pupils and this gives a good opportunity to assess understanding of concepts used, but is demanding of time. Such models are evaluated in Brennan and Resnick's survey of frameworks suitable for the assessment of interactive media (Brennan and Resnick, 2012), which presents several scenarios for assessment which would be useful to explore in classroom settings and with different software tools.

Other research into how to assess programming and computational thinking offers assessments where pupils are asked to modify, extend or correct errors in a program (Werner et al., 2012b) or involves peer instruction (Simon and Cutts, 2012), which helps pupils to articulate their (mis)understandings and can give a better account of understanding of computing concepts than looking at the final outcome alone.

Assessment frameworks to support the new Computing programme of study have been developed (e.g. Dorling and Walker, 2014) but are arguably less useful when assessing extended projects such as computer games, and ignore the design process

and the development of learning/thinking skills which are an important feature of constructionist learning activities.

10.10 Concluding remarks

This thesis has explored the introduction of a unit of work in which Year 9 pupils created a computer game for the first time as part of the Key Stage 3 ICT curriculum.

In considering pupils' perceptions of the process of their learning and the games they made, this research makes a useful contribution to discussions surrounding the pedagogy of computer game authoring in mainstream school settings.

The findings show how, as they made their games, pupils learned some basic game design and programming concepts, developed their ability to think computationally and gained an awareness of the 'constructedness' of digital media, becoming producers of software for the first time.

And importantly, they valued doing so:

> JG: It makes you think more than the plain work that everybody does … this makes you think really hard about what you are doing. At the end of it you can play your game, and you think, 'Oh, I've made this!'
>
> JB: I feel [it's] … more of an achievement, 'cos we have our proper, finished product … it's not just a piece of paper, it's, like, *more*. I feel really happy with it, that we've managed to do something like that!

For any educator those are precious words, because they articulate pupils' pride in their achievements and their identification as creators and thinkers. They are also a striking testament to the importance of Seymour Papert's vision for the proper use of computers in education.

References

AARDMAN & NOMINET TRUST 2014. 'Shaun the Sheep Game Academy' [Online]. Available: http://shaunsgameacademy.co.uk [Accessed 23/08/14].

AARSETH, E. 2003. 'Playing research: methodological approaches to game analysis.' In: MILES, A. (ed.) *Proceedings of the 5th International Digital Arts and Culture Conference*. Melbourne, 19-23 May. Melbourne, Australia: RMIT University. 1-7.

ABC ONLINE 2004. 'Seymour Papert: Sunday Profile'. *ABC Local Radio Sunday Profile* [Online]. Available: http://www.abc.net.au/sundayprofile/stories/s1144341.htm [Accessed 14/01/14].

ACKERMANN, E. 2004. 'Constructing knowledge and transforming the world.' In: TOKORO, M. & STEELS, L. (eds.) *A learning zone of one's own: sharing representations and flow in collaborative learning environments*. Washington, USA: IOS Press.

ADAMS, J. & WEBSTER, A. 2012. 'What do students learn about programming from game, music video and storytelling projects?' In: SMITH KING, L., MUSICANT, D., CAMP, T. & TYMANN, P. (eds.) *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*. Raleigh, USA, 29 February-3 March. New York, USA: ACM. 643-648.

ADIPRANATA, Y. & ADIPRANATA, R. 2010. 'Teaching object oriented programming course using cooperative learning method based on game design and visual object oriented environment.' In: MAHADEVAN, V. & TOMAR, G. (eds.) *Proceedings of the 2nd International Conference on Education Technology and Computer*, *Vol. 2*. Shanghai, China, 22-24 June. Los Alamitos, USA: IEEE. 355-359.

ADOBE 2007. *Adobe Flash Professional*. [Computer program]. Adobe Systems.

AHMADI, N. & JAZAYERI, M. 2014. 'Analysing the learning process in online educational game design: a case study.' *Proceedings of the 23rd Australasian Software Engineering Conference*. Milson's Point, Australia, 7-10 April. Los Alamitos, USA: IEEE. 84-93.

AMMA 2011. 'Scratch or Game Maker?' [Online]. AmmA Centre. Available: http://ammacentre.org/blog/?p=383 [Accessed 06/10/12].

APPS FOR GOOD 2013. Apps for Good [Online]. Available: http://www.appsforgood.org/ [Accessed 23/07/13].

AQA 2005. *Information & Communication Technology A: higher tier paper*. AQA.

AQA 2012a. *GCSE Computer Science specification*. AQA.

AQA 2012b. *GCSE ICT specification*. AQA.

AQA 2012c. *GCSE in Computer Science component 1: practical programming. Specimen candidate booklet: scenario 2 - computer gaming application*. AQA.

AQA 2012d. 'KS3/4 Computing: let's make a game of it - Game Maker.' [Online]. Available: http://www.aqa.org.uk/professional-development/course-details?meta_E=SCSCGCSEKS34ComputingLetsmakeagameofITGamemaker [Accessed 06/10/12].

BALDWIN, L. & KULJIS, J. 2000. 'Visualisation techniques for learning and teaching programming.' *Journal of Computing and Information Technology*, 8 (4), 285-291.

BANAJI, S. & BURN, A. 2007. 'Creativity through a rhetorical lens: implications for schooling, literacy and media education.' *Literacy*, 41 (2), 62-70.

BARNES, T., POWELL, E., CHAFFIN, A. & LIPFORD, H. 2008. 'Game2Learn: improving the motivation of CS1 students.' In: FOSTER, J., NORDLINGER, J. & YOUNG, M. (eds.) *Proceedings of the 3rd International Conference on Game Development in Computer Science Education*. Miami, USA, 28 February-3 March. New York, USA: ACM. 1-5.

BARR, D., HARRISON, J. & CONERY, L. 2011. 'Computational thinking: a digital age skill for everyone.' *Learning and Leading with Technology.* Mar/Apr 2011, 21-23. ISTE.

BAYTAK, A. 2009. 'An investigation of the artifacts, outcomes and processes of constructing computer games about environmental science in a fifth grade science classroom.' PhD thesis, Pennsylvania State University, USA.

BAYTAK, A. & LAND, S. 2010. 'A case study of educational game design by kids and for kids.' *Procedia - Social and Behavioral Sciences*, 2 (2), 5242-5246.

BAYTAK, A. & LAND, S. 2011a. 'Advancing elementary school girls' programming through game design.' *International Journal of Gender, Science and Technology,* 3 (1). [Online]. Available: http://genderandset.open.ac.uk/index.php/genderandset/article/viewArticle/88.

BAYTAK, A. & LAND, S. 2011b. 'An investigation of the artifacts and process of constructing computer games about environmental science in a fifth grade classroom.' *Educational Technology Research and Development,* 59 (6), 765-782.

BAYTAK, A., LAND, S. & SMITH, B. 2011. 'Children as educational computer game designers: an exploratory study.' *The Turkish Online Journal of Educational Technology,* 10 (4), 84-92.

BAYTAK, A., LAND, S., SMITH, B. & PARK, S. 2008. 'An exploratory study of kids as educational game designers.' In: SIMONSON, M. (ed.) *Proceedings of the 31st Annual Convention of the Association for Educational Communications and Technology.* Orlando, USA, 6-9 November. Bloomington, USA: AECT Publications. 39-47.

BAZELEY, P. 2007. *Qualitative data analysis with NVivo*. London: Sage Publications Ltd.

BCS 2012. *Draft programme of study for ICT*. BCS.

BEAVIS, C. 2013. 'Multiliteracies in the wild: learning from computer games.' In: MERCHANT, G., GILLEN, J., MARSH, J. & DAVIES, J. (eds.) *Virtual literacies - interactive spaces for children and young people.* New York, USA: Routledge. 57-74.

BEAVIS, C. & O'MARA, J. 2010. 'Computer games - pushing at the boundaries of literacy.' *Australian Journal of Language and Literacy*, 33 (1), 65-76.

BEAVIS, C., O'MARA, J. & MCNEICE, L. (eds.) 2012. *Digital games: literacy in action*. Kent Town, Australia: Wakefield Press.

BECKETT, H. 2013. 'Green light for our inner geeks.' *Times Educational Supplement, STEM*. 15/03/2013 TSL Education Ltd.

BECTA 2006. *Computer games in education project report*. Becta.

BEGEL, A. & KLOPFER, E. 2004. 'StarLogo TNG: an introduction to game development.' [Online]. Available: http://research.microsoft.com/en-us/um/people/abegel/papers/starlogo-tng.pdf [Accessed 16/03/14].

BEN-ARI, M. 2001. 'Constructivism in computer science education.' *Journal of Computers in Mathematics and Science Teaching,* 20 (1), 45-73.

BERGLUND, A., DANIELS, M. & PEARS, A. 2006. 'Qualitative research projects in computing education research: an overview.' In: YOUNG, A. & TOLHURST, D. (eds.) *Proceedings of the 7th Australasian Conference on Computing Education*. Hobart, Australia, 16-19 January. Darlinghurst, Australia: ACS. 25-33.

BERMINGHAM, S., CHARLIER, N., DAGNINO, F., DUGGAN, J., EARP, J., KIILI, K., LUTS, E., VAN DER STOCK, L., & WHITTON, N. 2013. 'Approaches to collaborative game making for fostering 21st century skills.' In: ESCUDEIRO, P. & VAZ DE CARVALHO, C. (eds.) *Proceedings of the 7th European Conference on Games-Based Learning*. Porto, Portugal, 3-4 October. Reading: ACPI. 45-52.

BEYNON, M. 2009. 'Constructivist computer science education reconstructed.' *Innovation in Teaching and Learning in Information and Computer Sciences,* 8 (2), 73-90.

BEYNON, M. & HARFIELD, A. 2010. 'Constructionism through construal by computer.' *Paper presented at Constructionism 2010*. Paris, France, 16-21 August.

BEYNON, M. & ROE, C. 2004. 'Computer support for constructionism in context.' In: LOOI, C., SUTINEN, E., SAMPSON, D., AEDO, I., UDEN, L. & KAHKONEN, E. (eds.) *Proceedings of the International Conference on Advanced Learning Technologies*. Joensuu, Finland, 30 August-1 September. Los Alamitos, USA: IEEE. 216-220.

BIGGS, J. 1979. 'Individual differences in study processes and the quality of learning outcomes.' *Higher Education*, 8 (4), 381-394.

BIGGS, J. 2003. SOLO taxonomy [Online]. Available: http://www.johnbiggs.com.au/academic/solo-taxonomy/ [Accessed 04/08/13].

BIGGS, J. & COLLIS, K. 1982. *Evaluating the quality of learning - the SOLO taxonomy*. New York, USA: Academic Press.

BOBER, M. 2010. *Games-based experiences for learning*. Futurelab.

BOWKER, A. 1998. *Flowol 2*. [Computer program]. Keep I.T. Easy.

BOWKER, A. 2005. *Flowol 3*. [Computer program]. Keep I.T. Easy.

BRABAND, C. & DAHL, B. 2009. 'Analyzing CS competencies using the SOLO taxonomy.' *Proceedings of the Conference on Innovation and Technology in Computer Science Education*. Paris, France, 6-9 July. New York, USA: ACM. 1-1.

BRAGGE, J. & STORGARDS, J. 2007. 'Profiling academic research on digital games using text mining tools.' *Proceedings of the 3rd International Conference of the Digital Games Research Association*. Tokyo, Japan, 24-28 September. DiGRA. 714-729.

BRAY, O. 2011. *Playful learning: computer games in education*. Microsoft Education.

BRENNAN, K. 2013a. 'Learning computing through creating and connecting.' *Computer*, 46 (9), 52-59.

BRENNAN, K. 2013b. 'Best of both worlds: issues of structure and agency in computational creation, in and out of school.' PhD thesis, Massachusetts Institute of Technology, USA.

BRENNAN, K., CHUNG, M. & HAWSON, J. 2011. *Creative computing - a design-based introduction to computational thinking.* MIT Media Lab, USA.

BRENNAN, K. & RESNICK, M. 2012. 'Using artifact-based interviews to study the development of computational thinking in interactive media design.' *Paper presented at the American Educational Research Association Conference*. Vancouver, Canada, 13-17 April.

BRENNAN, K. & RESNICK, M. 2013. 'Imagining, creating, playing, sharing, reflecting: how online community supports young people as designers of interactive media.' In: LAVIGNE, N. & MOUZA, C. (eds.) *Emerging technologies for the classroom: a learning sciences perspective*. New York, USA: Springer. 253-268.

BRUCKMAN, A., EDWARDS, E., ELLIOTT, J., JENSEN, C. 2000. 'Uneven achievement in a constructionist learning environment.' In: FISHMAN, B. & O'CONNOR-DIVELBISS, S. (eds.) *Proceedings of the 4th International Conference of the Learning Sciences*. Ann Arbor, USA, 14-17 June. Mahwah, USA: Lawrence Erlbaum Associates. 157-163.

BRYANT, L., DOWNES, S., TWIST, J., PRENSKY, M., FACER, K., DUMBLETON, T. & LEY, D. 2007. *Emerging technologies for learning (Volume 2)*. Becta.

BUCKINGHAM, D. 2003. *Media education: literacy, learning and contemporary culture*. Cambridge: Polity Press.

BUCKINGHAM, D. & BURN, A. 2007a. 'Game literacy in theory and practice.' *Journal of Educational Multimedia and Hypermedia*, 16 (3), 323-349.

BUCKINGHAM, D. & BURN, A. 2007b. 'Making games: game design and media literacy.' *English, Drama, Media*, 8. NATE.

BUCKINGHAM, D., SEFTON-GREEN, J. & WILLETT, R. 2003. *Shared spaces: informal learning and digital cultures*. Institute of Education, University of London.

BULFIN, S., HENDERSON, M. & JOHNSON, N. 2013. 'Examining the use of theory within educational technology and media research.' *Learning, Media and Technology*, 38 (3), 337-344.

BURN, A. 2007. ‘‘Writing’ computer games: game literacy and new-old narratives.’ *Educational Studies in Language and Literature*, 7 (4), 45-67.

BURTOFT, M., GARVEY, P., HENDERSON, E., KELLY, K. & LOCKHART, T. 2008. *Year 7 Smart Skills Builder ICT.* Cambridge: Smart Learning Ltd.

CARBONARO, M., CUTUMISU, M., DUFF, H., GILLIS, S., ONUCZKO, C., SIEGEL, J., SCHAEFFER, J., SCHUMACHER, A., SZAFRON, D. & WAUGH, K. 2008. ‘Interactive story authoring: a viable form of creative expression for the classroom.’ *Computers & Education*, 51 (2), 687–707.

CARBONARO, M., CUTUMISU, M., MCNAUGHTON, M., ONUCZKO, C., ROY, T., SCHAEFFER, J., SZAFRON, D., GILLIS, S. & KRATCHMER, S. 2005. ‘Interactive story writing in the classroom: using computer games.’ *Proceedings of the Digital Games Research Association International Conference*. Vancouver, Canada, 16-20 June. DiGRA. 323-338.

CARBONARO, M., SZAFRON, D., CUTUMISU, M. & SCHAEFFER, J. 2010. ‘Computer-game construction: a gender-neutral attractor to Computing Science.’ *Computers & Education*, 55 (3), 1098-1111.

CAS 2008a. Computing at School [Online]. Available: http://www.computingatschool.org.uk/ [Accessed 02/11/09].

CAS 2008b. *Rationale for a GCSE in Computing*. CAS.

CAS 2012a. *Computer Science: a curriculum for schools*. CAS.

CAS 2012b. *A curriculum framework for Computer Science and Information Technology.* CAS.

CAS 2014. *Switched On*. Autumn issue 2014. CAS.

CAVALLO, D., PAPERT, S. & STAGER, G. 2004. ‘Climbing to understanding: lessons from an experimental learning environment for adjudicated youth.’ In: KAFAI, Y., SANDOVAL, W., ENYEDY, N., NIXON, A. & HERRERA, F. (eds.) *Proceedings of the 6th International Conference of the Learning Sciences*. Santa Monica, USA, 22-26 June. Mahwah, USA: Lawrence Erlbaum Associates. 113-120.

CHAFFIN, A., DORAN, K., HICKS, D. & BARNES, T. 2009. ‘Experimental evaluation of teaching recursion in a video game.’ In: SPENCER, S., DAVIDSON, D., FULLERTON, T. & SCHRIER, K. (eds.) *Proceedings of the SIGGRAPH Symposium on Video Games.* New Orleans, USA, 3-7 August. New York, USA: ACM. 79-86.

CHAMILLARD, A. 2006. ‘Introductory game creation: no programming required.’ In: BALDWIN, D., TYMANN, P., HALLER, S. & RUSSELL, I. (eds.) *Proceedings of the SIGSCE Technical Symposium on Computer Science Education*. Houston, USA, 1-5 March. New York, USA: ACM. 515-519.

CHAN, C., TSUI, M., CHAN, M. & HONG, J. 2002. ‘Applying the Structure of Observed Learning Outcomes (SOLO) taxonomy on students' learning outcomes: an empirical study.’ *Assessment and Evaluation in Higher Education*, 27 (6), 511-527.

CHENG, G. 2009. ‘Using game making pedagogy to facilitate student learning of interactive multimedia.’ *Australasian Journal of Educational Technology*, 25 (2), 204-220.

CLAYPOOL, K. & CLAYPOOL, M. 2005. ‘Teaching software engineering through game design.’ In: CUNHA, J., FLEISCHMAN, W., PROULX, V. & LOURENÇO, J. (eds.) *Proceedings of the 10th Annual Conference on Innovation and Technology in Computer Science Education*. Caparica, Portugal, 26-29 June. New York, USA: ACM. 123-127.

COHEN, L., MANION, L. & MORRISON, K. 2007. *Research methods in education*. 6th edition. Abingdon: Routledge.

CONNOLLY, T., BOYLE, E., MACARTHUR, E., HAINEY, T. & BOYLE, J. 2012. ‘A systematic literature review of empirical evidence on computer games and serious games.’ *Computers & Education*, 59 (2), 661-686.

CONSALVO, M. & DUTTON, N. 2006. ‘Game analysis: developing a methodological toolkit for the qualitative study of games.’ *Game Studies,* 6 (1). [Online]. Available: http://gamestudies.org/0601/articles/consalvo_dutton [Accessed 16/01/15].

COOPER, S., DANN, W. & PAUSCH, R. 1999. *Alice 2*. [Computer program]. Carnegie Mellon University, USA.

COY, S. 2013. ‘Kodu Game Lab - a few lessons learned.’ *Crossroads,* 19 (4), 44-47. ACM.

CRESWELL, J. W. 2007. *Qualitative inquiry and research design: choosing among five approaches*. 2nd edition. Thousand Oaks, USA: Sage.

CRESWELL, J. W. 2014. *Research design: qualitative, quantitative and mixed methods approaches*. Thousand Oaks, USA: Sage.

CUTTS, Q., ESPER, S., FECHO, M., FOSTER, S. & SIMON, B. 2012. ‘The abstraction transition taxonomy: developing desired learning outcomes through the lens of situated cognition.’ In: CLEAR, A., SANDERS, K. & SIMON, B. (eds.) *Proceedings of the International Workshop on Computing Education Research*. Auckland, New Zealand, 10-12 September. New York, USA: ACM. 63-70.

CUTTS, Q., ESPER, S. & SIMON, B. 2011. ‘Computing as the 4th 'R': a general education approach to computing education.’ In: SANDERS, K., CASPERSEN, M. & CLEAR, A. (eds.) *Proceedings of the International Computing Education Research Conference*. Providence, USA, 8-9 August. New York, USA: ACM. 133-138.

DAGDILELIS, V., SATRATZEMI, M. & EVANGELIDIS, G. 2004. ‘Introducing secondary education students to algorithms and programming.’ *Education and Information Technologies*, 9 (2), 159-173.

DALAL, N., DALAL, P., KAK, S., ANTONENKO, P. & STANSBERRY, S. 2009. ‘Rapid digital game creation for broadening participation in computing and fostering crucial thinking skills.’ *International Journal of Social and Humanistic Computing,* 1 (2), 123-137.

DALAL, N., KAK, S. & SOHONI, S. 2012. ‘Rapid digital game creation for learning object-oriented concepts.’ In: COHEN, E. & BOYD, E. (eds.) *Proceedings of Informing*

Science & IT Education Conference. Montreal, Canada, 22-27 June. Santa Rosa, USA: Informing Science Institute. 237-247.

DALY, T. 2009. 'Using introductory programming tools to teach programming concepts: a literature review.' *The Journal for Computing Teachers*. Autumn issue. 1-6. ISTE.

DAVIS, R., KAFAI, Y., VASUDEVAN, V. & EUNKYOUNG, L. 2013. 'The Education Arcade: crafting, remixing, and playing with controllers for Scratch games.' In: HOURCADE, J., SAWHNEY, N. & REARDON, E. (eds.) *Proceedings of the 12th International Conference on Interaction Design and Children.* New York, USA, 24-27 June. New York, USA: ACM. 439-442.

DCSF 2008a. *The framework for secondary ICT*. DCSF.

DCSF 2008b. *ICT subject leader development materials Summer 2008: handout 4.5 ICT unit plan 3 template - sequencing*. DCSF.

DE FREITAS, S. 2006. *Learning in immersive worlds: a review of game-based learning*. JISC.

DEECD 2010. *'The impact of web 2.0 technologies in the classroom - KnowledgeBank: Next Generation research report Kodu excerpt.*' Melbourne, Australia: DEECD.

DENNER, J. 2007. 'The Girls Creating Games program: an innovative approach to integrating technology into middle school.' *Meridian Middle School Computer Technologies Journal*, 10 (1).

DENNER, J. & WERNER, L. 2011. 'Measuring computational thinking in middle school using game programming.' *Paper presented at the American Educational Research Association Conference*, New Orleans, USA, 8-12 April.

DENNER, J., WERNER, L., BEAN, S. & CAMPE, S. 2005. 'The Girls Creating Games program: strategies for engaging middle school girls in information technology.' *Frontiers: A Journal of Women's Studies*, 26 (1), 90-98.

DENNER, J., WERNER, L. & ORTIZ, E. 2012. 'Computer games created by middle school girls: can they be used to measure understanding of computer science concepts?' *Computers & Education*, 58 (1), 240–249.

DFE 1995. *Information Technology in the National Curriculum*. HMSO.

DFE 2013a. *Assessing without levels* [Online]. Available: http://www.education.gov.uk/schools/teachingandlearning/curriculum/nationalcurriculum2014/a00225864/assessing-without-levels [Accessed 29/07/13].

DFE 2013b. *Consultation on the order for replacing the subject of ICT with Computing: government response*. DfE.

DFE 2013c. *The National Curriculum in England: Computing - programmes of study - Key Stages 3 and 4*. DfE.

DFE 2013d. *The National Curriculum in England: framework document for consultation*. DfE.

DFEE 1999. *The National Curriculum handbook for secondary teachers in England.* DfEE.

DFES 2002a. *Key Stage 3 National Strategy framework for teaching ICT capability: Years 7, 8 and 9*. DfES.

DFES 2002b. *Key Stage 3 National Strategy sample teaching unit 7.6*. DfES.

DFES 2003a. *Key Stage 3 National Strategy for ICT: progression into and through Year 9 - case studies*. DfES.

DFES 2003b. *Key Stage 3 National Strategy ICT sample teaching unit 8.5*. DfES.

DFES 2003c. *Key Stage 3 National Strategy ICT Year 9 case studies: unit 9.1*. DfES.

DFES 2004. *Pedagogy and practice: teaching and learning in secondary schools. Unit 17: developing effective learners*. DfES.

DILLON, T. 2004. *Adventure games for learning and storytelling*. Futurelab.

DISESSA, A. 1997. 'Twenty reasons why you should use Boxer (instead of Logo).' In: TURCSÁNYI-SZABÓ, M. (ed.) *Proceedings of the 6th European Logo Conference*. Budapest, 20-23 August. Budapest, Hungary: John von Neumann Computer Society & Eötvös Loránd University. 7-27.

DORAN, K., BOYCE, A., FINKELSTEIN, S. & BARNES, T. 2012. 'Outreach for improved student performance: a game design and development curriculum.' *Proceedings of the 17th Annual Conference on Innovation and Technology in Computer Science Education*. Haifa, Israel, 3-5 July. New York, USA: ACM. 209-214.

DORLING, M. & WALKER, M. 2014. *Computing progression pathways*. CAS.

DOWNS, Y. 2010. 'Transcription tales or let love's labour not be lost.' *International Journal of Research and Method in Education*, 33 (1). 101-112.

DOWNES, T. 1999. 'Playing with computing technologies in the home.' *Education and Information Technologies*, 4 (1), 65-79.

DOYLE, S. 2004. *ICT framework solutions Year 7: student book*. Cheltenham: Nelson Thornes.

DU BOULAY, B. 1986. 'Some difficulties of learning to program.' *Journal of Educational Computing Research*, 2 (1), 57-73.

E-LINE MEDIA 2013. *Gamestar Mechanic* [Online]. Available: www.gamestarmechanic.com [Accessed 20/04/13].

EDEXCEL 2006. *D202 Multimedia summative project brief - Crack the Code*. Edexcel.

EDEXCEL 2009. *D205 Games authoring summative project brief - Disaster Strikes!* Edexcel.

EDEXCEL 2012a. *Certificate in Digital Applications specification*. Pearson Education Ltd.

EDEXCEL 2012b. *GCSE Computer Science specification*. Edexcel.

EDEXCEL 2012c. *GCSE in ICT specification*. Edexcel.

EDWARDS, A. & WESTGATE, D. 1994. *Investigating classroom talk*. 2nd edition. London: The Falmer Press.

EGENFELDT-NIELSEN, S. 2006. 'Overview of research on the educational use of video games.' *Digital Kompetanse*, 1 (3), 184-213.

ELSPA 2006. *Unlimited learning: computer and video games in the learning landscape.* ELSPA.

ENTERBRAIN 2005. *RPG Maker XP*. [Computer program]. Degica.

EOW, Y., WAN ALI, W., MAHMUD, R. & BAKI, R. 2010. 'Computer games development and the appreciative learning approach in enhancing students' creative perception.' *Computers & Education*, 54 (1), 146-161.

ESPER, S., FOSTER, S. & GRISWOLD, W. 2013. 'On the nature of fires and how to spark them when you're not there.' In: CAMP, T., TYMANN, P., DOUGHERTY, J. & NAGEL, K. (eds.) *Proceedings of the SIGCSE Technical Symposium on Computer Science Education.* Denver, USA, 6-9 March. New York, USA: ACM. 305-310.

EVERS, C. & WU, E. 2007. 'On generalising from single case studies: epistemological reflections.' In: BRIDGES, D. & SMITH, R. (eds.) *Philosophy, methodology and educational research.* Malden: Blackwell. 199-213.

FACER, K., ULICSAK, M. & SANDFORD, R. 2007. 'Can computer games go to school?' In: BRYANT, L., DOWNES, S., TWIST, J., PRENSKY, M., FACER, K., DUMBLETON, T. & LEY, D. *Emerging technologies for learning (Volume 2)*. Becta. 47-63.

FELICIA, P. (ed.) 2011. *Handbook of research on improving learning and motivation through educational games: multidisciplinary approaches.* Hershey, USA: Information Science Reference.

FELICIA, P. (ed.) 2013. *Developments in current game-based learning design and deployment*. Hershey, USA: IGI Global.

FERDIG, R. & BOYER, J. 2007. 'Can game development impact academic achievement?' *THE Journal* [Online]. Available: http://thejournal.com/articles/2007/10/25/can-game-development-impact-academic-achievement.aspx [Accessed 05/08/13].

FIEGE, M. 2011. 'Teaching programming concepts by building games.' MSc dissertation, Delft University of Technology, Netherlands.

FINCHER, S. 2006. *Studying programming*. Basingstoke: Palgrave Macmillan.

FLETCHER, G. & LU, J. 2009. 'Human computing skills: rethinking the K-12 experience.' *Communications of the ACM*, 52 (2), 23-25.

FLUCK, A. & MEIJERS, M. 2006. 'Game making for students and teachers from isolated areas' [Online]. Available:

http://www.une.edu.au/simerr/pages/projects/79gamemaking.php. [Accessed 05/08/13].

FONSECA, C., KOZBERG, G., TEMPEL, M., SOPRUNOV, S., YAKOVLEVA, E., REGGINI, H., RICHARDSON, J., ALMEIDA, M. & CAVALLO, D. 1999. *Logo philosophy and implementation*. LCSI.

FORSTER, T. 2006. 'Other worlds and game creation.' [Online]. Available: http://tonyforster.blogspot.co.uk/search?q=Other+worlds+and+game+creation. [Accessed 05/0814].

FOWLER, A. & CUSACK, B. 2011. 'Enhancing introductory programming with Kodu Game Lab: an exploratory study.' In: MANN, S. & VERHAART, M. (eds.) *Proceedings of the 2nd Annual Conference of Computing and Information Technology Education and Research in New Zealand*. Rotorua, 5-8 July. Hamilton, New Zealand: CITRENZ. 69-79.

FRANZ, G. & PAPERT, S. 1988. 'Computer as material: messing about with time.' *Teachers' College Record*, 89 (3), 408-17.

FRISTOE, T., DENNER, J., MACLAURIN, M., MATEAS, M. & WARDRIP-FRUIN, N. 2011. 'Say it with systems: expanding Kodu's expressive power through gender-inclusive mechanics.' *Proceedings of the 6th International Conference on Foundations of Digital Games*. Bordeaux, France, 28 June-1 July. New York, USA: ACM. 227-234.

FROMME, J. & UNGER, A. (eds.) 2012. *Computer games and new media cultures: a handbook of digital games studies*. Dordrecht, Netherlands: Springer.

FRYDENBERG, M. 2013. 'Creating a collaborative learning community in the CIS Sandbox.' *Interactive Technology and Smart Education*, 10 (1), 49-62.

FULLER, U., JOHNSON, G., AHONIEMI, T., CUKIERMAN, D., HERNAN-LOSADA, I., JACKOVA, J., LAHTINEN, E., LEWIS, T., THOMPSON, D., RIEDESEL, C. & THOMPSON, E. 2007. 'Developing a computer science-specific learning taxonomy.' *Proceedings of the 12th Annual Conference on Innovation and Technology in Computer Science Education.* Dundee, Scotland, 23-27 June. New York, USA: ACM. 152-170.

FURBER, S. 2012. *Shut down or restart - the way forward for computing in UK schools*. The Royal Society.

FURLONGER, C. & HAYWOOD, S. 2004. *Teaching the National ICT Strategy at Key Stage 3: a practical guide*. London: David Fulton.

GAMBLE, P. 2009. 'Game Maker in schools.' *Game Maker Technology Magazine*. Issue 16, 9-11.

GAMES FOR CHANGE 2013. 'Games for Change' [Online]. Available: http://www.gamesforchange.org/ [Accessed 23/07/13].

GAMES, I. A. 2008a. 'Gamestar Mechanic: reflections on the design and research of a game about game design.' [Online]. Available: http://meaningfulplay.msu.edu/proceedings2008/mp2008_paper_79.pdf [Accessed 12/12/14].

GAMES, I. A. 2008b. ‘Three dialogs: a framework for the analysis and assessment of twenty-first-century literacy practices, and its use in the context of game design within Gamestar Mechanic.’ *E-learning & Digital Media*, 5 (4), 396-417.

GAMES, I. A. 2010. ‘Gamestar Mechanic: learning a designer mindset through communicational competence with the language of games.’ *Learning, Media and Technology,* 35 (1), 31-52.

GEE, J. P. 2003a. ‘What video games have to teach us about learning and literacy.’ *Computers in Entertainment*, 1 (1), 1-4.

GEE, J. P. 2003b. *What video games have to teach us about learning and literacy*. Basingstoke: Palgrave Macmillan.

GERRING, J. 2007. *Case study research: principles and practice*. Cambridge: Cambridge University Press.

GIBBS, G. 2002. *Qualitative data analysis: explorations with NVivo*. Maidenhead: Oxford University Press.

GILES, J., BEARD, S. & STREET, S. 2008. *ICT 4 life*. Harlow: Pearson Education Ltd.

GOOD, J. 2011. ‘Learners at the wheel: novice programming environments come of age.’ *International Journal of People-Oriented Programming*, 1 (1), 1-24.

GOOD, J., HOWLAND, K. & NICHOLSON, K. 2010. ‘Young people’s descriptions of computational rules in role-play games: an empirical study.’ In: HUNDHAUSEN, C., PIETRIGA, E., DIAZ, P. & ROSSON, M. (eds.) *Proceedings of the IEEE Symposium on Visual Languages and Human Centric Computing*. Madrid, Spain, 21-25 September. Los Alamitos, USA: IEEE. 67-74.

GOOD, J. & ROBERTSON, J. 2004. ‘Computer games authored by children - a multi-perspective evaluation.’ *Proceedings of the Conference on Interaction Design and Children*. Baltimore, USA, 1-3 June. New York, USA: ACM. 123-124.

GOOD, J. & ROBERTSON, J. 2006a. ‘A framework for learner-centred design with children.’ *International Journal of Artificial Intelligence in Education,* 16 (4), 381-413.

GOOD, J. & ROBERTSON, J. 2006b. ‘Learning and motivational affordances in narrative-based game authoring.’ In: BRNA, P. (ed.) *Proceedings of the 4th International Conference for Narrative and Interactive Learning Environments*. Edinburgh, Scotland, 8-11 August. Edinburgh: NILE. 37-51.

GOOD, J., ROMERO, P., DU BOULAY, B., ROBERTSON, J., REID, H. & HOWLAND, K. 2007. ‘Authoring as acting: exploring embodied interaction in game authoring environments for children: Full research report.’ Swindon: ESRC.

GOOGLE 2011. ‘Exploring computational thinking’ [Online]. Available: http://www.google.com/edu/computational-thinking/index.html [Accessed 22/09/12].

GOSLING, D. 2007. ‘Micro-power relations between teachers and students using five perspectives on teaching in higher education.’ [Online]. Available: http://www.davidgosling.net/userfiles/micro%20power%20relations%20isl%202007.pdf Accessed [10/08/14].

GOVE, M. 2012a. *Written ministerial statement by Michael Gove on Information and Communication Technology (ICT)*. [Online]. Available: https://www.gov.uk/government/speeches/written-ministerial-statement-by-michael-gove-on-information-and-communication-technology-ict [Accessed 13/12/14].

GOVE, M. 2012b. *Michael Gove speech at the BETT show 2012*. [Online]. Available: https://www.gov.uk/government/speeches/michael-gove-speech-at-the-bett-show-2012 [Accessed 13/12/14].

GOVENDER, I., GOVENDER, D., HAVENGA, M., MENTZ, E., BREED, B., DIGNUM, F. & DIGNUM, V. 2014. 'Increasing self-efficacy in learning to program: exploring the benefits of explicit instruction for problem solving.' *The Journal for Transdisciplinary Research in Southern Africa*, 10 (1), 187-200.

GRANT, L. 2010. *Developing the home-school relationship using digital technologies*. Futurelab.

GROSS, P., HERSTAND, M., HODGES, J. & KELLEHER, C. 2010. 'A code reuse interface for non-programmer middle school students.' In: RICH, C., YANG, Q., CAVAZZA, M. & ZHOU, M. (eds.) *Proceedings of the International Conference on Intelligent User Interfaces*. Hong Kong, 7-10 February. New York, USA: ACM. 219-228.

GROVER, S. 2011. 'Robotics and engineering for middle and high school students to develop computational thinking.' *Paper presented at the Annual Meeting of the American Educational Research Association,* New Orleans, USA, 7-11 April.

GUIMARAES, M. & MURRAY, M. 2008. 'An exploratory overview of teaching computer game development.' *Journal of Computing Sciences in Colleges*, 24 (1), 144-149.

GUZDIAL, M. 2009. 'Question everything: how we teach intro CS is wrong.' *Computing Education Blog* [Online]. Available: http://computinged.wordpress.com/2009/10/02/question-everything-how-we-teach-intro-cs-is-wrong/ [Accessed 07/0714].

HABGOOD, J. 2006. 'Compulsory game development for everyone.' [Online]. Available: http://www.gamasutra.com/view/news/101319/Education_Feature_Compulsory_Game_Development.php. [Accessed 05/08/14].

HABGOOD, J., NIELSEN, N. & RIJKS, M. 2010. *The game maker's companion*. New York, USA: Apress.

HABGOOD, J. 2013. *Game Maker: Studio* [PowerPoint presentation]. *Windows 8 and Windows Phone 8 Game Development in Education event*. Birmingham City University, England, 26 March. Available: http://www.hiddenlevel.com [Accessed 15/02/15].

HADJERROUIT, S. 2008. 'Using a learner-centred approach to teach ICT in secondary schools: an exploratory study.' *Issues in Informing Science and Information Technology*, 5, 233-259.

HAGUE, C. & WILLIAMSON, B. 2009. *Digital participation, digital literacy, and school subjects*. Futurelab.

HAMMOND, M. 2004. ‘The peculiarities of teaching Information and Communication Technology as a subject: a study of trainee and new ICT teachers in secondary schools.’ *Technology, Pedagogy and Education*, 13 (1), 29-42.

HAMMOND, M., YOUNIE, S., WOOLLARD, J., CARTWRIGHT, V. & BENZIE, D. 2009. *What does our past involvement with computers in education tell us? A view from the research community*. Coventry: Warwick University Press.

HAREL, I. 1991. *Children designers: interdisciplinary constructions for learning and knowing mathematics in a computer-rich school*. Norwood, USA: Ablex.

HAREL, I. & PAPERT, S. (eds.) 1991a. *Constructionism: research reports and essays 1985-1990*. Norwood, USA: Ablex.

HAREL, I. & PAPERT, S. 1991b. ‘Software design as a learning environment.’ In: HAREL, I. & PAPERT, S. (eds.) *Constructionism*. Norwood, USA: Ablex. 41-84.

HAREL CAPERTON, I. 2010. ‘Toward a theory of game media literacy: playing and building as reading and writing.’ *International Journal of Gaming and Computer-Mediated Simulations*, 2 (1), 1-16.

HAREL CAPERTON, I., SULLIVAN, S., OLIVER, A., LOWENSTEIN, D., BATTJER, M., ROSENFELT, R., LA PORTA, A., MINNIGERODE, L. & REYNOLDS, R. 2006. *Globaloria World Wide Workshop*. [Online]. Available: http://www.globaloria.org/ [Accessed 19/07/13].

HARTEVELD, C., SMITH, G., CARMICHAEL, G., GEE, E. & STEWART-GARDINER, C. 2014. ‘A design-focused analysis of games teaching computer science.’ *Paper presented at the Games, Learning and Society Conference.* University of Wisconsin-Madison, USA, 11-13 June.

HAWKINS, W. & HEDBERG, J. 1986. ‘Evaluating Logo: use of the SOLO taxonomy.’ *Australian Journal of Educational Technology*, 2 (2), 103-109.

HAY, K. & BARAB, S. 2001. ‘Constructivism in practice: a comparison and contrast of apprenticeship and constructionist learning environments.’ *The Journal of the Learning Sciences*, 10 (3), 281-322.

HAYES, E. & GAMES, I. 2008. ‘Making computer games and design thinking.’ *Games and Culture*, 3 (3-4), 309-332.

HENDERSON, C., YERUSHALMI, E., KUO, V., HELLER, K. & HELLER, P. 2007. ‘Physics faculty beliefs and values about the teaching and learning of problem solving. II. Procedures for measurement and analysis.’ *Physical Review Special Topics: Physics Education Research*, 3 (2), 020110-1 – 020110-12.

HERNANDEZ, C., SILVA, L., SEGURA, R., SCHIMIGUEL, J., LEDON, M., BEZERRA, L. & SILVEIRA, I. 2010. ‘Teaching programming principles through a game engine.’ *CLEI Electronic Journal*, 13 (2), 1-8.

HERRIG, B. 2013. ‘Get your head in the game: digital game-based learning with Game Maker.’ In: BAEK, Y. & WHITTON, N. (eds.) *Cases on digital game-based learning: methods, models and strategies*. Hershey, USA: IGI Global. 228-239.

HOGANSON, K. 2010. ‘Teaching programming concepts with Game Maker.’ *Journal of Computing Sciences in Colleges,* 26 (2), 181-188.

HOWLAND, J., LAFFEY, J. & ESPINOSA, L. 1997. ‘A computing experience to motivate children to complex performances.’ *Journal of Computing in Childhood Education*, 8 (4), 291-311.

HOWLAND, K., GOOD, J. & DU BOULAY, B. 2008. ‘A game creation tool which supports the development of writing skills: interface design considerations.’ *Proceedings of the 5th International Conference on Narrative and Interactive Learning Environments*. Edinburgh, Scotland, 5-8 August. Edinburgh: NILE. 23-29.

HOWLAND, K., GOOD, J., & DU BOULAY, B. 2013. ‘Narrative Threads: a tool to support young people in creating their own narrative-based computer games.’ In: PAN, Z., CHEOK, A., MULLER, W., IURGEL, I., PETTA, P., URBAN, B. (eds.) *Transactions on edutainment X: lecture notes on computer science* Vol. 7775. Heidelberg, Germany: Springer. 122-145.

HOWLAND, K., GOOD, J. & NICHOLSON, K. 2009. ‘Language-based support for computational thinking.’ In: DELINE, R., MINAS, M. & ERWIG, M. (eds.) *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing*. Corvallis, USA, 20-24 September. Piscataway, USA: IEEE. 147-150.

HOWLAND, K., GOOD, J. & ROBERTSON, J. 2006. ‘Script Cards: a visual programming language for games authoring by young people.’ In: GRUNDY, J. & HOWSE, J. (eds.) *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing*. Brighton, England, 4-8 September. Los Alamitos, USA: IEEE. 181-186.

HWANG, G., HUNG, C. & CHEN, N. 2014. ‘Improving learning achievements, motivations and problem-solving skills through a peer assessment-based game development approach.’ *Educational Technology Research and Development*, 62 (2), 129-145.

HWANG, G. & WU, P. 2012. Advancements and trends in digital game-based learning research: a review of publications in selected journals from 2001 to 2010. *British Journal of Educational Technology*, 43 (1), E6-E10.

ICAA 2001. *GCSE Information and Communication Technology specification (G43)*. ICAA.

IMMERSIVE EDUCATION 2007. *MissionMaker*. [Computer program]. Immersive Education.

JENKINS, T. 2002. ‘On the difficulty of learning to program.’ [Online]. Available: http://www.psy.gla.ac.uk/~steve/localed/jenkins.html [Accessed 16/01/15].

JEWITT, C. 2008. *The visual in learning and creativity: a review of the literature.* Arts Council England.

JONASSEN, D. 1996. *Computers in the classroom: mindtools for critical thinking*. Columbus, USA: Merrill/Prentice Hall.

JONES, D. & WILSON, D. 2008. *Pixel8 Game Maker tutorials*. teach-ict.com Ltd.

JUNG, J. & PARK, H. 2009. ‘Learning by doing via game making.’ In: GIBSON, D. & BAEK, Y. (eds.) *Digital simulations for improving education: learning though artificial teaching environments*. Hershey, USA: Information Science Reference. 394-406.

JUUL, J. 2003. ‘The game, the player, the world: looking for a heart of gameness.’ In: COPIER, M. & RAESSENS, J. (eds.) *Proceedings of Level Up: Digital Games Research Conference*. Utrecht, Netherlands, 4-6 November. Utrecht: Utrecht University. 30-45.

KAFAI, Y. 1995. *Minds in play: computer game design as a context for children’s learning*. Hillsdale, USA: Lawrence Erlbaum Associates.

KAFAI, Y. 1996. ‘Learning design by making games: children's development of design strategies in the creation of a complex computational artifact.’ In: KAFAI, Y. & RESNICK, M. (eds.) *Constructionism in practice: designing, thinking and learning in a digital world*. Mahwah, USA: Lawrence Erlbaum Associates. 71-96.

KAFAI, Y. 1998. ‘Video game designs by girls and boys: variability and consistency of gender differences.’ In: CASSELL, J. & JENKINS, H. (eds.) *From Barbie to Mortal Kombat*: *gender and computer games.* Cambridge, USA: MIT Press. 90-117.

KAFAI, Y. 2001. ‘The educational potential of electronic games: from games-to-teach to games-to-learn.’ *Paper presented* at *Playing by the Rules: The Cultural Policy Challenges of Video Games Conference*. University of Chicago, USA, 26-27 October.

KAFAI, Y. 2006a. ‘Constructionism.’ In: SAWYER, R. (ed.) *The Cambridge handbook of the learning sciences*. New York, USA: Cambridge University Press. 35-46.

KAFAI, Y. 2006b. ‘Playing and making games for learning: instructionist and constructionist perspectives for game studies.’ *Games and Culture*, 1 (1), 36-40.

KAFAI, Y. & BURKE, Q. 2014. ‘Mindstorms 2: children, programming, and computational participation.’ [Online]. Available: http://constructionism2014.ifs.tuwien.ac.at/papers/2.6_3-8530.pdf [Accessed 15/02/15].

KAFAI, Y., BURKE, W. & FIELDS, D. 2009a. ‘What videogame making can teach us about access and ethics in participatory culture.’ [Online]. Available: http://www.digra.org/wp-content/uploads/digital-library/09287.14579.pdf [Accessed 16/01/15].

KAFAI, Y., CHING, C. & MARSHALL, S. 1997. ‘Children as designers of educational multimedia software.’ *Computers & Education*, 29 (2-3), 117-126.

KAFAI, Y. & HAREL, I. 1991. ‘Learning through design and teaching: exploring social and collaborative aspects of constructionism.’ In: HAREL, I. & PAPERT, S. (eds.) *Constructionism*. Norwood, USA: Ablex. 85-106.

KAFAI, Y. & PEPPLER, K. 2011. ‘Youth, technology and DIY: developing participatory competencies in creative media production.’ *Review of Research in Education*, 35 (1), 89-119.

KAFAI, Y. & PEPPLER, K. 2012. ‘Developing gaming fluencies with Scratch: realising game design as an artistic process.’ In: STEINKUEHLER, C., SQUIRE, K. & BARAB, S. (eds.) *Games, learning and society*. New York, USA: Cambridge University Press. 355-380.

KAFAI, Y., PEPPLER, K. & CHAPMAN, R. (eds.) 2009b. *The Computer Clubhouse: constructionism and creativity in youth communities*. New York, USA: Teachers College Press.

KAFAI, Y. & RESNICK, M. (eds.) 1996a. *Constructionism in practice: designing, thinking and learning in a digital world*. Mahwah, USA: Lawrence Erlbaum Associates.

KAFAI, Y. & RESNICK, M. 1996b. 'Introduction.' In: KAFAI, Y. & RESNICK, M. (eds.) *Constructionism in practice: designing, thinking and learning in a digital world*. Mahwah, USA: Lawrence Erlbaum Associates.

KANE, L., ANTON, G., BERGER, W., SHAPIRO, B. & SQUIRE, K. 2012. 'Studio K: a game design curriculum for computational thinking.' [Online]. Available: http://learning.wrexham.gov.uk/pluginfile.php/4024/mod_resource/content/0/Kodu_CurriculumDoc.pdf [Accessed 16/01/15].

KAZIMOGLU, C., KIERNAN, M., BACON, L. & MACKINNON, L. 2012. 'A serious game for developing computational thinking and learning introductory computer programming.' *Procedia - Social & Behavioural Sciences,* 47, 1991-1999.

KE, F. 2009. 'A qualitative meta-analysis of computer games as learning tools.' In: FERDIG, R. (ed.) *Handbook of research on effective electronic gaming in education*. Hershey, USA: IGI Global. 1-32.

KE, F. 2014. 'An implementation of design-based learning through creating educational computer games: a case study on mathematics learning during design and computing.' *Computers & Education*, 73, 26-39.

KELLEHER, C. & PAUSCH, R. 2005. 'Lowering the barriers to programming: a taxonomy of programming enivironments and languages for novice programmers.' *ACM Computing Surveys*, 37 (2), 83-137.

KELLEHER, C. & PAUSCH, R. 2007. 'Using storytelling to motivate programming.' *Communications of the ACM*, 50 (7), 58-64.

KEMP, P. 2014. *Computing in the National Curriculum: a guide for secondary teachers*. CAS.

KENNEWELL, S. & MORGAN, A. 2006. 'Factors influencing learning through play in ICT settings.' *Computers and Education*, 46 (3), 265-279.

KENNEWELL, S., TANNER, H., JONES, S. & BEAUCHAMP, G. 2007. 'Analysing the use of interactive technology to implement interactive teaching.' *Journal of Computer Assisted Learning,* 24 (1), 61-73.

KHAN ACADEMY 2012. 'Computer programming' [Online]. Available: https://www.khanacademy.org/cs [Accessed 02/02/14].

KINNUNEN, P. & SIMON, B. 2012. 'Phenomenography and grounded theory as research methods in the computing education research field.' *Computer Science Education*, 22 (2), 199-218.

KIRK, O. 2006. 'Study of a game engine for the Nintendo Game Boy Advance.' Department of Computer Science, University of Copenhagen. [Online]. Available: http://image.diku.dk/projects/media/kirk.06.pdf [Accessed 16/01/15].

KIRRIEMUIR, J. & MCFARLANE, A. 2004. *Literature review in games and learning*. Futurelab.

KIRSCHNER, P., SWELLER, J. & CLARK, R. 2006. 'Why minimal guidance during instruction does not work: an analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching.' *Educational Psychologist*, 41 (2), 75-86.

KLOPFER, E., OSTERWEIL, S. & SALEN, K. 2009. *Moving learning games forward: obstacles, opportunities and openness*. Cambridge, USA: The Education Arcade, MIT.

KOH, K., BASAWAPATNA, A., BENNETT, V. & REPENNING, A. 2010. 'Towards the automatic recognition of computational thinking for adaptive visual language learning.' In: HUNDHAUSEN, C., PIETRIGA, E., DÍAZ, P. & ROSSON, M. (eds.) *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing*. Madrid, Spain, 21-25 September. Los Alamitos, USA: IEEE. 59-66.

KONZACK, L. 2002. 'Computer game criticism: a method for computer game analysis.' In: MAYRA, F. (ed.) *Proceedings of the Computer Games and Digital Cultures Conference*. Tampere, Finland, 6-8 June. Tampere: Tampere University Press. 89-100.

KUATO 2013. *Hakitzu*. Kuato Studios. [Online]. Available: http://www.kuatostudios.com [Accessed 13/09/14].

KURLAND, M., CLEMENT, C., MAWBY, R. & PEA, R. 1987. 'Mapping the cognitive demands of learning to program.' In: PERKINS, D., LOCHEAD, J. & BISHOP, J. (eds.) *Thinking: progress in research and teaching.* Hillsdale, USA: Lawrence Erlbaum. 333-358.

KURUVADA, P., ASAMOAH, A., DALAL, N. & KAK, S. 2010a. 'Learning computational thinking from rapid digital game creation.' *Proceedings of the 2nd Annual Conference on Theoretical and Applied Computer Science*. Stillwater, USA, 5 November. Stillwater: Oklahoma State University. 31-36.

KURUVADA, P., ASAMOAH, A., DALAL, N. & KAK, S. 2010b. 'The use of rapid game creation to learn computational thinking.' [Online]. Available: http://arxiv.org/ftp/arxiv/papers/1011/1011.4093.pdf [Accessed 16/01/15].

KUTNICK, P., SEBBA, J., BLATCHFORD, P., GALTON, M. & THORP, J. 2005. *The effects of pupil grouping: literature review. Research report RR688*. Nottingham: DFES Publications.

LARSEN MCCLARTY, K., ORR, A., FREY, P., DOLAN, R., VASSILEVA, V. & MCVAY, A. 2012. *A literature review of gaming in education*. Pearson.

LAHTINEN, E., ALA-MUTKA, K. & JARVINEN, H. 2005. 'A study of the difficulties of novice programmers.' In: CUNHA, J., FLEISCHMAN, W., PROULX, V. & LOURENCO, J. (eds.) *Proceedings of the 10th Conference on Innovation and Technology in Computer Science Education*. Caparica, Portugal, 26-29 June. New York, USA: ACM. 14-18.

LAVE, J. & WENGER, E. 1991. *Situated learning: legitimate peripheral participation*. Cambridge: Cambridge University Press.

LAVONEN, J., MEISALO, V., LATTU, M. & SUTINEN, E. 2003. 'Concretising the programming task: a case study in a secondary school.' *Computers & Education*, 40 (2), 115-135.

LAWSON, M. 2010. 'An investigation into the process of teacher assessment of ICT capability in a sample of schools in the North West of England.' EdD thesis, University of Huddersfield, England.

LEAFLINE 2003. *Digit Strategy: Unit 7.6 Ghost Train*. Leafline.

LTS 2009. *Curriculum for excellence. Technologies: experiences and outcomes*. Learning and Teaching Scotland.

LTS 2010. *Games-based learning* [Online]. Available: http://www.ltscotland.org.uk/ictineducation/gamesbasedlearning/index.asp [Accessed 03/04/10].

LEWIS C. 2011. 'Is pair programming more effective than other forms of collaboration for young students?' *Computer Science Education*, 21 (2), 105-134.

LI, Q. 2010. 'Digital game building: learning in a participatory culture.' *Educational Research*, 52 (4), 427-443.

LIFELONG KINDERGARTEN GROUP 2013. *Scratch website* [Online]. Available: http://scratch.mit.edu/ [Accessed 11/05/13].

LIM, E. & BINTI MD SABRI, M. 2013. 'Learning history through computer game authoring.' *Proceedings of the 8th International Conference on Computer Science & Education*. Colombo, Sri Lanka, 26-28 April. Los Alamitos, USA: IEEE. 746-750.

LIN, J., YEN, L., YANG, M. & CHEN, C. 2005. 'Teaching computer programming in elementary schools: a pilot study.' *Paper presented at the National Educational Computing Conference*. Philadelphia, USA, 27-30 June.

LISTER, R., SIMON, B., THOMPSON, E., WHALLEY, J. & PRASAD, C. 2006. 'Not seeing the forest for the trees: novice programmers and the SOLO taxonomy.' In: DAVOLI, R., GOLDWEBER, M. & SALOMONI, P. (eds.) *Proceedings of the Conference on Innovation and Technology in Computer Science Education*. Bologna, Italy, 26-28 June. New York, USA: ACM. 118-122.

LIU, C., CHENG, Y. & HUANG, C. 2011. 'The effect of simulation games on the learning of computational problem solving.' *Computers & Education*, 57 (3), 1907-1918.

LIVINGSTONE, I. & HOPE, A. 2011. *Next Gen: transforming the UK into the world's leading talent hub for the video games and visual effects industries*. Nesta.

LOVELESS, A. & WEGERIF, R. 2004. 'Unlocking creativity with ICT.' In: FISHER, R. & WILLIAMS, M. (eds.) *Unlocking creativity: teaching across the curriculum*. Abingdon: David Fulton Publishers. 92-102.

LUCKIN, R. 2010. *Re-designing learning contexts: technology-rich, learner-centred ecologies.* Abingdon: Routledge.

LUCKIN, R., BLIGH, B., MANCHES, A., AINSWORTH, S., CROOK, C. & NOSS, R. 2012. *Decoding learning: the proof, promise and potential of digital education*. Nesta.

MACKLIN, C. 2010. *Activate!* [Online]. Available: http://www.activategames.org/ [Accessed 19/07/13].

MACKLIN, C. & SHARP, J. 2012. ‘‘Freakin’ hard’: game design and issue literacy.’ In: STEINKUEHLER, C., SQUIRE, K. & BARAB, S. (eds.) *Games, learning and society*. New York, USA: Cambridge University Press. 381-402.

MACLURE, M. 2006. ‘‘A demented form of the familiar’: postmodernism and educational research.’ *Journal of Philosophy of Education*, 40 (2), 223-239.

MACROMEDIA 2004. *Macromedia Fireworks MX 2004*. [Computer program]. Macromedia.

MADILL, L. & SANFORD, K. 2009. ‘Video-game creation as a learning experience for teachers and students.’ In: FERDIG, R. (ed.) *Handbook of research on effective electronic gaming in education.* Hershey, USA: Information Science Reference. 1257-1272.

MAGUIRE, P., MAGUIRE, R., HYLAND, P. & MARSHALL, P. 2014. ‘Enhancing collaborative learning using pair programming: who benefits?’ *All Ireland Journal of Teaching and Learning in Higher Education*, 6 (2), 1411-14125.

MALONEY, J., PEPPLER, K., KAFAI, Y., RESNICK, M. & RUSK, N. 2008. ‘Programming by choice: urban youth learning programming with Scratch.’ In: DOUGHERTY, J., RODGER, S., FITZGERALD, S. & GUZDIAL, M. (eds.) *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*. Portland, USA,12-15 March. New York, USA: ACM. 367-371.

MCDOUGALL, A. & BOYLE, M. 2004. ‘Student strategies for learning computer programming: implications for pedagogy in informatics.’ *Education and Information Technologies*, 9 (2), 109-116.

MCINERNEY, C. 2010. ‘Having fun with computer programming and games: teacher and student experiences.’ In: HROMKOVIC, J., KRALOVIC, R. & VAHRENHOLD, J. (eds.) *Proceedings of the 4th International Conference on Informatics in Secondary Schools*. Zurich, Switzerland, 13-15 January. Berlin, Germany: Springer. 136-142.

MEERBAUM-SALANT, O., ARMONI, M. & BEN-ARI, M. 2010. ‘Learning computer science concepts with Scratch.’ In: CASPERSEN, M., CLANCY, M. & SANDERS, K. (eds.) *Proceedings of the International Computing Education Research Conference*. Aarhus, Denmark, 9-10 August. New York, USA: ACM. 69-76.

MEERBAUM-SALANT, O., ARMONI, M. & BEN-ARI, M. 2011. ‘Habits of programming in Scratch.’ In: RÖßLING, G., NAPS, T. & SPANNAGEL, C. (eds.) *Proceedings of the Conference on Innovation and Technology in Computer Science Education*. Darmstadt, Germany, 27-29 June. New York, USA: ACM. 168-172.

MEIJERS, M. 2012. *ICT Mindtools* [Online]. Available: http://ictmindtools.net/ [Accessed 28/04/12].

MENDELSOHN, P., GREEN, T. & BRNA, P. 1990. ‘Programming languages in education.’ In: HOC, J., GREEN, T., SAMURÇAY, R. & GILMORE, D. (eds.) *Psychology of programming.* London: Academic Press Ltd. 175-200.

MERCHANT, G., GILLEN, J., MARSH, J. & DAVIES, J. (eds.) 2013. *Virtual literacies: interactive spaces for children and young people*. New York, USA: Routledge.

MERRIAM, S. 1998. *Qualitative research and case study applications in education*. 2nd edition. San Fransisco, USA: Jossey-Bass.

MERRIAM, S. 2009. *Qualitative research: a guide to design and implementation*. San Fransisco, USA: Jossey-Bass.

MICROSOFT RESEARCH 2009. *Kodu*. [Computer program]. Microsoft Research.

MILLWOOD, R. 2009. *A short history off-line*. Becta.

MINNIGERODE, L. & REYNOLDS, R. 2013. 'Don't give up: a case study on girls and video game design.' *Learning Landscapes*, 6 (2), 283-302.

MISHRA, P. & KOEHLER, M. 2006. 'Technological pedagogical content knowledge: a framework for teacher knowledge.' *Teachers' College Record*, 108 (6), 1017-1054.

MITCHELL, A. & SAVILL-SMITH, C. 2004. *The use of computer and video games for learning: a review of the literature*. Learning and Skills Development Agency.

MOLINS-RUANO, P., SEVILLA, C., SANTINI, S., HAYA, P., RODRIGUEZ, P. & SACHA, G. 2014. 'Designing videogames to improve students' motivation.' *Computers in Human Behaviour*, 31, 571-579.

MORGAN, A. & KENNEWELL, S. 2005. 'The role of play in the pedagogy of ICT.' *Education and Information Technologies*, 10 (3), 177-188.

MOZILLA 2013a. *Maker Party* [Online]. Available: https://webmaker.org/en-US/party [Accessed 29/09/13].

MOZILLA 2013b. *OpenBadges* [Online]. Available: http://openbadges.org/ [Accessed 03/02/14].

MOZILLA 2013c. *Webmaker* [Online]. Mozilla. Available: https://webmaker.org/ [Accessed 27/08/13].

MURATET, M., TORGUET, P., VIALLET, F. & JESSEL, J. 2011. 'Experimental feedback on Prog&Play: a serious game for programming practice.' *Computer Graphics Forum*, 30 (1), 61-73.

MURNANE, J. 2010. *Programming languages for beginners*. Saarbrücken, Germany: Lamber Academic Publishing.

MURNANE, J. & MCDOUGALL, A. 2006. 'Bad computer science in beginners programming courses: 'considered harmful'? A case study of the Tufts graphical programming language.' In: WATSON, D. & BENZIE, D. (eds.) *Proceedings of the IFIP Joint Conference - Imagining the Future for ICT and Education*. Alesund, Norway, 26-30 June. Alesund: Alesund University College. 1-12.

NAACE 2012. *Draft Naace curriculum framework: Information and Communication Technology (ICT) Key Stage 3*. NAACE.

NATIONAL RESEARCH COUNCIL 2009. *Report of a workshop on the scope and nature of computational thinking*. National Research Council.

NATVIG, L. & LINE, S. 2004. 'Age of Computers: game-based teaching of computer fundamentals.' In*:* BOYLE, R., CLARK, M. & KUMAR, A. (eds.) *Proceedings of the Conference on Innovation and Technology in Computer Science Education*. Leeds, England, 28-30 June. New York, USA: ACM. 107-111.

NAVARRETE, C. & MINNIGERODE, L. 2013. 'Exploring 21st century learning: game design and creation, the students' experience.' [Online]. Available: http://www.worldwideworkshop.org/pdfs/Globaloria_UTAustinResearch_Sept2013.pdf [Accessed 09/08/14].

NELSON, M. 2001. *Robocode*. [Computer game]. SourceForge.

NESTA 2012. *Digital Makers programme* [Online]. Available: http://www.nesta.org.uk/areas_of_work/public_services_lab/digital_education/assets/features/digital_makers [Accessed 22/11/12].

NESTA, NOMINET TRUST & MOZILLA 2013. *Make things do stuff* [Online]. Available: http://makethingsdostuff.co.uk/ [Accessed 30/09/13].

NESTA 2014. 'How can teachers prepare for the new computing curriculum?' [Online]. Available: http://www.nesta.org.uk/blog/how-can-teachers-prepare-new-computing-curriculum [Accessed 29/07/14].

NORMAN, D. 1983. 'Some observations on mental models.' In: GENTNER, D. & STEVENS, A. (eds.) *Mental models*. Hillsdale, USA: Lawrence Erlbaum Associates. 7-14.

NORTHCOTT, B. & MILISZEWSKA, I. 2008. 'Facilitating creativity in vocational computer game development courses: the role of technology and pedagogy.' In: USKOV, V. (ed.) *Proceedings of the 11th International Conference on Computers and Advanced Technology in Education*. Crete, Greece, 29 September-1 October. Calgary, Canada: Acta Press. 402-407.

NOSS, R., COX, R., LAURILLARD, D., LUCKIN, R., PLOWMAN, L., SCANLON, E. & SHARPLES, M. 2012. *System upgrade: realising the vision for UK education*. London Knowledge Lab.

O'MARA, J. & RICHARDS, J. 2012. 'A blank slate: using Game Maker to create computer games.' In: BEAVIS, C., O'MARA, J. & MCNEICE, L. (eds.) *Digital games: literacy in action*. Kent Town, Australia: Wakefield Press. 57-64.

O'NEIL, H., WAINESS, R. & BAKER, E. 2005. 'Classification of learning outcomes: evidence from the computer games literature.' *The Curriculum Journal*, 16 (4), 455 – 474.

OCR 2006. *GCSE ICT Paper 1 (Foundation Tier)*. OCR.

OCR 2009a. *GCSE ICT Unit B065: Coding a solution - specimen controlled assessment material*. OCR.

OCR 2009b. *GCSE ICT specification*. OCR.

OCR 2010. *GCSE in Computing specification*. OCR.

OCR 2011. *GCSE in Computing specification.* 2nd edition. OCR.

OCR 2012a. *Cambridge National Certificate in ICT Unit R008: Introduction to computer programming - model assignment assessment material*. OCR.

OCR 2012b. *OCR GCSE ICT Unit B064: Creative use of ICT - controlled assessment task - candidate style answers*. OCR.

OCR 2013. *Unit B064 Creative use of ICT - unit recording sheet*. OCR.

OFCOM 2013. *Children and parents: media use and attitudes report*. Ofcom.

OFSTED 2008. *Using data, improving schools*. OFSTED.

OFSTED 2009. *The importance of ICT: Information and Communication Technology in primary and secondary schools, 2005-2008*. OFSTED.

OFSTED 2011. *ICT in schools 2008-11*. OFSTED.

OSTROVSKY, I. 2009. *Robozzle*. [Computer game]. Available: http://www.robozzle.com [Accessed 08/02/15].

OVERMARS, M. 2003. *Game Maker Community forum* [Online]. Available: http://gmc.yoyogames.com/ [Accessed 14/07/12].

OVERMARS, M. 2004. 'Teaching computer science through game design.' *Computer,* 37 (4), 81-83.

OVERMARS, M. 2015. *Email to Claire Johnson*, 22 February.

OWSTON, R., WIDEMAN, H., RONDA, N. & BROWN, C. 2009. 'Computer game development as a literacy activity.' *Computers & Education*, 53 (2), 977-989.

PAPASTERGIOU, M. 2009. 'Digital game-based learning in high school Computer Science education: impact on educational effectiveness and student motivation.' *Computers & Education*, 52 (1), 1-12.

PAPERT, S. 1970. 'Teaching children thinking.' In: SCHEEPMAKER, B. & ZINN, K. (eds.) *Proceedings of the IFIP World Congress on Computers and Education*. Amsterdam, 24-28 August. Groningen, Netherlands: Wolters Noordhoff. 73-78.

PAPERT, S. 1980a. 'Computer-based microworlds as incubators for powerful ideas.' In: TAYLOR, R. (ed.) *The computer in the school: tutor, tool, tutee*. New York, USA: Teachers College Press. 203-210.

PAPERT, S. 1980b. *Mindstorms - children, computers, and powerful ideas*. New York, USA: Basic Books.

PAPERT, S. 1980c. 'Teaching children thinking.' In: TAYLOR, R. (ed.) *The computer in the school: tutor, tool, tutee*. New York, USA: Teachers College Press. 161-176.

PAPERT, S. 1980s. 'Constructionism vs instructionism.' [Online]. Available: http://www.papert.org/articles/const_inst/const_inst1.html [Accessed 26/09/11].

PAPERT, S. 1984. ‘Computer as mudpie.’ *Classroom Computer Learning*, 4 (6), 36-38.

PAPERT, S. 1986. ‘Constructionism: a new opportunity for elementary science education.’ *Proposal to the National Science Foundation*. MIT Media Lab.

PAPERT, S. 1990a. ‘A critique of technocentrism in thinking about the school of the future.’ *Epistemology and Learning memo no. 2*. MIT Media Lab.

PAPERT, S. 1990b. ‘Introduction.’ In: HAREL, I. (ed.) *Constructionist learning*: *a fifith anniversary collection of papers reflecting research reports, projects in progress, and essays by the Epistemology & Learning Group*. MIT Media Lab. 7-15.

PAPERT, S. 1991a. ‘Forward.’ In: HAREL, I. *Children designers: interdisciplinary constructions for learning and knowing mathematics in a computer-rich school.* Norwood, USA: Ablex Publishing Corporation. xi-xiii.

PAPERT, S. 1991b. ‘Situating constructionism.’ In: HAREL, I. & PAPERT, S. (eds.) *Constructionism*. Norwood, USA: Ablex Publishing Corporation. 1-11.

PAPERT, S. 1993. *The children's machine: rethinking school in the age of the computer*. New York, USA: Basic Books.

PAPERT, S. 1994. ‘Keynote speech.’ *Technology & Learning Conference.* Dallas, USA, 28 October. National School Boards Association.

PAPERT, S. 1996a. *The connected family: bridging the digital generation gap*. Marietta, USA: Longstreet Press.

PAPERT, S. 1996b. ‘An exploration in the space of mathematics educations.’ *International Journal of Computers for Mathematical Learning*, 1 (1), 95-123.

PAPERT, S. 1997. ‘Educational computing: how are we doing?’ *Technological Horizons in Education*, 24 (11), 78-80.

PAPERT, S. 1998a. ‘Child power: keys to the new learning of the digital century.’ *Colin Cherry memorial lecture on communication*. Imperial College, London, 2 June.

PAPERT, S. 1998b. ‘Does easy do it? Children, games, and learning.’ *Game Developer*, 5 (6), 87-88.

PAPERT, S. 1999a. ‘Eight big ideas behind the Constructionist Learning Lab.’ [Online]. Available: http://stager.org/articles/8bigideas.pdf [Accessed 16/01/15].

PAPERT, S. 1999b. ‘Introduction: What is Logo and who needs it?’ In: FONSECA, C., KOZBERG, G., TEMPEL, M., SOPRUNOV, S., YAKOVLEVA, E., REGGINI, H., RICHARDSON, J., ALMEIDA, M. & CAVALLO, D. (eds.) *Logo implementation and philosophy*. LCSI. iv-xvi.

PAPERT, S. 2001. ‘Project-based learning.’ [Online]. Available: http://www.edutopia.org/seymour-papert-project-based-learning#graph4. [Accessed 11/08/14].

PAPERT, S. 2002. ‘How to make writing ‘hard fun’.’ *Bangor Daily News*, 24/06/2002.

PAPERT, S., BOBROW, D. & FEURZEIG, W. 1967. *Logo*. [Computer program]. MIT.

PAPERT, S. & SOLOMON, C. 1971. *Twenty things to do with a computer: Logo memo no. 3*. MIT Artificial Intelligence Laboratory.

PARSONS, D. & HADEN, P. 2007. 'Programming osmosis: knowledge transfer from imperative to visual programming environments.' In: MANN, S. & BRIDGEMAN, N. (eds.) *Proceedings of the 20th Annual Conference of the National Advisory Committee on Computing Qualifications*. Nelson, New Zealand, 8-11 July. Hamilton, New Zealand: NACCQ. 209-215.

PASSEY, D. 2012. *Independent evaluation of the Little Big Planet 2 project in Wolverhampton's local education partnership schools: outcomes and impacts - final report*. Lancaster: Lancaster University.

PAYTON, S. & HAGUE, C. 2010. *Digital literacy professional development resource*. Futurelab.

PEA, R. 1983. 'Logo programming and problem solving.' *Proceedings of the AERA Symposium - Chameleon in the Classroom: Developing Roles for Computers*. Montreal, Canada, 11-15 April. New York, USA: Bank Street College of Education. 25-33.

PEA, R. & KURLAND, M. 1984. 'On the cognitive effects of learning computer programming.' *New Ideas in Psychology*, 2 (2), 137-168.

PEA, R. 1986. 'Language-independent conceptual "bugs" in novice programming.' *Journal of Educational Computing Research,* 2 (1) 25-36.

PELLETIER, C. 2005. 'Studying games in school: a framework for media education.' [Online]. Available: http://www.digra.org/wp-content/uploads/digital-library/06278.32248.pdf [Accessed 16/01/15].

PELLETIER, C. 2007. *Making Games: developing games authoring software for educational and creative use: full research report*. ESRC end of award report RES-328-25-0001. Swindon: ESRC.

PELLETIER, C., BURN, A. & BUCKINGHAM, D. 2010. 'Game design as textual poaching: media literacy, creativity and game-making.' *E-Learning and Digital Media*, 7 (1), 90-107.

PEPPLER, K. & KAFAI, Y. 2005. 'Creative coding: programming for personal expression.' [Online]. Available: http://download.scratch.mit.edu/CreativeCoding.pdf [Accessed 16/01/15].

PEPPLER, K. & KAFAI, Y. 2007a. 'What video game making can teach us about literacy and learning: alternative pathways into participatory culture.' In: AKIRA, B. (ed.) *Proceedings of the 3rd International Conference of the Digital Games Research Association*. Tokyo, Japan, 24-28 September. Tokyo: University of Tokyo. 369-376.

PEPPLER, K. & KAFAI, Y. 2007b. 'From SuperGoo to Scratch: exploring creative digital media production in informal learning.' *Learning, Media and Technology*, 32 (2), 149-166.

PEPPLER, K. & KAFAI, Y. 2010. 'Gaming fluencies: pathways into a participatory culture in a community design studio.' *International Journal of Learning and Media*, 1 (4), 1-14.

PERKINS, D., HANCOCK, C., HOBBS, R. MARTIN, F. & SIMMONS, R. 1986. 'Conditions of learning in novice programmers.' In: SOLOWAY, E. & SPOHRER, J. (eds.) 1989. *Studying the novice programmer.* Hillsdale, USA: Lawrence Erlbaum Associates. 261-279.

PERKOVIĆ, L., SETTLE, A., HWANG, S. & JONES, J. 2010. 'A framework for computational thinking across the curriculum.' In: AYFER, R., IMPAGLIAZZO, J. & LAXER, C. (eds.) *Proceedings of the 15th Annual Conference on Innovation and Technology in Computer Science Education*. Ankara, Turkey, 28-30 June. New York, USA: ACM. 123-127.

PEROTTA, C., FEATHERSTONE, G., ASTON, H. & HOUGHTON, E. 2013. *Game-based learning: latest evidence and future directions*. NFER.

PEYTON JONES, S. 2010. *Computing at school: the state of the nation*. CAS.

PEYTON JONES, S., HERBERT, A., BISHOP, C., BOND, K., LANGFIELD, S. & HUMPHREYS, S. 2007. *Computing at school - white paper*. CAS.

PIAGET, J. 1972. *The principles of genetic epistemology*. London: Routledge & Kegan Paul.

PIAGET, J. 1973. *To understand is to invent: the future of education*. New York, USA: Viking Press.

PIAGET, J. & INHELDER, B. 1969. *The psychology of the child*. London: Routledge & Kegan Paul.

PIVEC, M. (ed.) 2009. *Proceedings of the 3rd European Conference on Games Based Learning*. Graz, Austria, 12-13 October. Reading: Academic Publishing Limited.

PIVEC, M. & PIVEC, P. 2008. 'Games in schools: executive summary.' [Online]. Available: http://www.paulpivec.com/Games_in_Schools.pdf [Accessed 16/01/15].

PRENSKY, M. 2001. *Digital game-based learning*. New York, USA: McGraw-Hill.

PRENSKY, M. 2002. 'What kids learn that's positive from playing games.' [Online]. Available: http://www.marcprensky.com/writing/Prensky%20-%20What%20Kids%20Learn%20Thats%20POSITIVE%20From%20Playing%20Video%20Games.pdf [Accessed 16/01/15].

PRENSKY, M. 2008. 'Students as designers and creators of educational computer games: who else?' *British Journal of Educational Technology*, 39 (6), 1004-1019.

PRITCHARD, A. & WOOLLARD, J. 2010. *Psychology for the classroom: constructivism and social learning*. Abingdon: Routledge.

PROPP, V. 1968. *Morphology of the folktale*. 2nd edition. Austin, USA: University of Texas Press.

QCA 2000. *ICT at Key Stage 3: scheme of work units 1-15*. QCA.

QCA 2007a. *A framework for personal learning and thinking skills*. QCA.

QCA 2007b. *ICT: programme of study for Key Stage 3*. QCA.

QCA 2009. *Cross curriculum dimensions: a planning guide for schools*. QCA.

QCA/NAA 2008. *ICT Key Stage 3: sequencing instructions task support*. QCA.

QCA/RM 2003. *Key Stage 3 ICT test*. QCA.

QSR INTERNATIONAL 2008. *NVivo 8*. [Computer program]. QSR International Pty Ltd.

REEVES, B. 2008. *ICT Interact for KS3: pupil's book 3*. London: Hodder Education.

REPENNING, A. & IOANNIDOU, A. 2008. 'Broadening participation through scalable game design.' In: DOUGHERTY, J., RODGER, S., FITZGERALD, S. & GUZDIAL, M. (eds.) *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education.* Portland, USA, 12-15 March. New York, USA: ACM. 305-309.

REPENNING, A., WEBB, D. & IOANNIDOU, A. 2010. 'Scalable game design and the development of a checklist for getting computational thinking into public schools.' In: LEWANDOWSKI, G., WOLFMAN, S., CORTINA, T. & LOWENFELD WALKER, E. (eds.) *Proceedings of the 41st SIGCSE Technical Symposium on Computer Science Education*. Milwaukee, USA, 10-13 March. New York, USA: ACM. 265-269.

RESNICK, M., FLANAGAN, M., KELLEHER, C., MACLAURIN, M., OHSHIMA, Y., PERLIN, K. & TORRES, R. 2009a. 'Growing up programming: democratising the creation of dynamic, interactive media.' *Proceedings of the Conference on Computer Human Interaction*: *Extended Abstracts on Human Factors in Computing Systems*. Boston, USA, 4-9 April. New York, USA: ACM. 3293-3296.

RESNICK, M., MALONEY, J., MONROY-HERNANDEZ, A., RUSK, N., EASTMOND, E., BRENNAN, K., MILLNER, A., ROSENBAUM, E., SILVER, J., SILVERMAN, B. & KAFAI, Y. 2009b. 'Scratch: programming for all.' *Communications of the ACM*, 52 (11), 60-67.

RESNICK, M., MALONEY, J., RUSK, N., EASTMOND, E., MILLNER, A., SILVER, J., ROSENBAUM, E., BRENNAN, K. & MONROY-HERNÁNDEZ, A. 2003. *Scratch*. [Computer program]. MIT Media Lab, Lifelong Kindergarten Group.

REYNOLDS, R., SCIALDONE, M. & CAPERTON, I. 2010. 'Evidence of high school students' development of contemporary learning abilities in a game design program in rural West Virginia: Globaloria student case study series, pilot year 2.' [Online]. Available: http://www.worldwideworkshop.org/pdfs/Globaloria_Year2_RTC_CaseStudyReport_2_16.pdf [Accessed 16/01/15].

RIEBER, L. 2005. 'Mulitimedia learning in games, simulations and microworlds.' In: MAYER, R. (ed.) *The Cambridge handbook of multimedia learning*. New York, USA: Cambridge University Press. 549-568.

RIEL, M. 1998. 'Education in the 21st Century: Just-in-time learning or learning communities.' *Paper presented at the 4th Annual Conference of the ECSSR: Challenges of the Next Millennium - Education & Development of Human Resources*. Abu Dhabi, UAE, 24-26 May.

ROBERTSON, D. 2009. ‘The Games in School community of practice.’ [Online]. Available: http://games.eun.org/2009/06/the_games_in_school_community_1.html. [Accessed 06/08/14].

ROBERTSON, J. 2004. ‘An analysis of the narrative features of computer games authored by children.’ In: BRNA, P. (ed.) *Proceedings of the International Conference for Narrative and Interactive Learning Environments*. Edinburgh, Scotland, 6-9 August. Edinburgh: AACE. 33-41.

ROBERTSON, J. 2012. ‘Making games in the classroom: benefits and gender concerns.’ *Computers & Education*, 59 (2), 385-398.

ROBERTSON, J. 2013. ‘The influence of a game-making project on male and female learners’ attitudes to computing.’ *Computer Science Education*, 23 (1), 58-83.

ROBERTSON, J. & GOOD, J. 2004. ‘Children's narrative development through computer game authoring.’ *TechTrends*, 49 (5), 43-59.

ROBERTSON, J. & GOOD, J. 2005. ‘Story creation in virtual game worlds.’ *Communications of the ACM*, 48 (1), 61-65.

ROBERTSON, J. & GOOD, J. 2006. ‘Supporting the development of interactive storytelling skills in teenagers.’ In: PAN, Z., AYLETT, R., DIENER, H., JIN, X., GÖBEL, S. & LI, L. (eds.) *Technologies for e-learning and Digital Entertainment: Lecture Notes in Computer Science*, Vol. 3942. Berlin, Germany: Springer. 348-357.

ROBERTSON, J. & HOWELLS, C. 2008. ‘Computer game design: opportunities for successful learning.’ *Computers & Education*, 50 (2), 559-578.

ROBERTSON, J. & NICHOLSON, K. 2007. ‘Adventure Author: a learning environment to support creative design.’ In: SKOV, M. (ed.) *Proceedings of the 6th International Conference on Interaction Design and Children*. Aalborg, Denmark, 6-8 June. New York, USA: ACM. 37-44.

ROBINS, A., ROUNTREE, J. & ROUNTREE, N. 2003. ‘Learning and teaching programming: a review and discussion.’ *Computer Science Education*, 13 (2), 137-172.

RODRÍGUEZ, F., KERBY, N. & BOYER, K. 2013. ‘Informing the design of a game-based learning environment for Computer Science: a pilot study on engagement and collaborative dialogue.’ In: WALKER, E. & LOOI, C. (eds.) *Proceedings of the Workshops at the 16th International Conference on Artificial Intelligence in Education*. Vol. 9. Memphis, USA, 9-13 July. CEUR-WS.org. 30-39.

RUSK, N. 2009. *Scratch cards*. [Online]. Available: http://scratch.mit.edu/help/cards/ [Accessed 16/01/15].

RYLANDS, T. 2007. *More on Myst* [Online]. Available: http://www.timrylands.com/more-on-myst/ [Accessed 01/09/13].

SAELI, M., PERRENET, J., JOCHEMS, W. & ZWANEVELD, B. 2011. ‘Teaching programming in secondary school: a pedagogical content knowledge perspective.’ *Informatics in Education*, 10 (1), 73-88.

SAELI, M., PERRENET, J., JOCHEMS, W. & ZWANEVELD, B. 2012. 'Programming: teachers and pedagogical content knowledge in the Netherlands.' *Informatics in Education*, 11 (1), 81-114.

SAINES, G., ERICKSON, S. & WINTER, N. 2013. *CodeCombat* [Online]. Available: http://codecombat.com/ [Accessed 28/08/14].

SALDANA, J. 2011. *The coding manual for qualitative researchers*. London: Sage Publications Ltd.

SALEN, K. 2007. 'Gaming literacies: a game design study in action.' *Journal of Educational Multimedia and Hypermedia*, 16 (3), 301-22.

SALEN, K., TORRES, R., WOLOZIN, L., RUFO-TEPPER, R. & SHAPIRO, A. 2011. *Quest to Learn: developing the school for digital kids*. Cambridge, USA: The MIT Press.

SANDFORD, R., ULICSAK, M., FACER, K. & RUDD, T. 2006. *Teaching with games: using commercial off-the-shelf computer games in formal education*. Futurelab.

SANDFORD, R. & WILLIAMSON, B. 2005. *Games and learning*. Futurelab.

SANFORD, K. & MADILL, L. 2007a. 'Recognising new literacies: teachers and students negotiating the creation of video games in school.' *Proceedings of the Digital Games Research Association Conference*. Tokyo, Japan, 24-28 September. DiGRA. 583-589.

SANFORD, K. & MADILL, L. 2007b. 'Understanding the power of new literacies through video game play and design.' *Canadian Journal of Education*, 30 (2), 432-455.

SARGENT, R., RESNICK, M., MARTIN, F. & SILVERMAN, B. 1996. 'Building and learning with programmable bricks.' In: KAFAI, Y. & RESNICK, M. (eds.) *Constructionism in practice: designing, thinking and learning in a digital world*. Mahwah, USA: Lawrence Erlbaum Associates. 161-173.

SCAA 1995. *Key Stage 3 Information Technology - the new requirements*. SCAA.

SCHELHOWE, H. 2007. *Technologie, imagination und lernen: grundlagen für bildungsprozesse mit digitalen medien*. Münster, Germany: Waxmann.

SCHELHOWE, H. 2010. 'Using construction kits: just learning how to program a computer - or is there more educational benefit?' *Paper presented at the Digital Media and Learning Conference,* La Jolla, USA, 18-20 February.

SCHMIDT, E. 2011. 'Television and the internet: shared opportunity.' *James MacTaggart memorial lecture*. [Online]. Available: www.geitf.co.uk/sites/default/files/geitf/GEITF_MacTaggart_2011_Eric_Schmidt.pdf [Accessed 16/01/15].

SCHMITZ, B., CZAUDERNA, A., KLEMKE, R. & SPECHT, M. 2011. 'Game based learning for computer science education.' In: VAN DER VEER, G., SLOEP, P. & VAN EEKELEN, M. (eds). *Proceedings of the Computer Science Education Research Conference*. Heerlen, Netherlands, 7-8 April. Heerlen: Open Universiteit. 81-86.

SCOTT, J. 2011. *Starting from Scratch - an introduction to computing science*. The Royal Society of Edinburgh.

SCRATCHED 2009. *ScratchEd* [Online]. Available: http://scratched.gse.harvard.edu/ Accessed [07/09/2014]

SEEHORN, D., CAREY, S., FUSCHETTO, B., LEE, I., MOIX, D., O'GRADY-CUNNIFF, D., BOUCHER OWENS, B., STEPHENSON, C. & VERNO, A. 2011. *CSTA K-12 Computer Science standards*. ACM.

SEFTON-GREEN, J. 2013. *Mapping digital makers: a review exploring everyday creativity, learning lives and the digital*. Nominet Trust.

SELBY, C. 2013. 'Computational thinking: the developing definition.' *Paper presented at the Conference on Innovation and Technology in Computer Science Education*. Canterbury, England, 1-3 July.

SHACKLETON, P., O'CONNOR, M. & TATNALL, A. 1997. 'Visual programming environments in information systems curricula.' In: SUTTON, D. (ed.) *Proceedings of the Australasian Conference on Information Systems*. Adelaide, Australia, 29 September-2 October. Adelaide: University of South Australia. 1-12.

SHAW, E., BOEHM, Z., PENWALA, H. & KIM, J. 2012. 'GameMath! Embedding secondary mathematics into a game making curriculum.' *Proceedings of the American Society of Engineering Education Conference*. San Antonio, USA, 10-13 June. Washington, USA: ASEE. 5045-5057.

SHEARD, J., CARBONE, A., LISTER, R., SIMON, B., THOMPSON, E. & WHALLEY, J. 2008. 'Going SOLO to assess novice programmers.' In: AMILLO, J., LAXER, C., MENASALVAS RUIZ, E. & YOUNG, A. (eds.) *Proceedings of the Conference on Innovation and Technology in Computer Science Education*. Madrid, Spain, 30 June-2 July. New York, USA: ACM. 209-213.

SHEARD, J., SIMON, S., HAMILTON, M. & LONNBERG, J. 2009. 'Analysis of research into the teaching and learning of programming.' In: CLANCY, M., CASPERSEN, M. & LISTER, R. (eds.) *Proceedings of the International Computing Education Research Workshop*. Berkeley, USA, 10-11 August. New York, USA: ACM. 93-104.

SILVERMAN, D. 2011. *Interpeting qualitative data: a guide to the principles of qualitative research*. 4th edition. London: Sage.

SILVERMAN, D. 2013. *Doing qualitative research*. 4th edition. Los Angeles, USA: Sage.

SIMON, B. & CUTTS, Q. 2012. 'Peer instruction: a teaching method to foster deep understanding.' *Communications of the ACM*, 55 (2), 27-29.

SMITH, D. 2000. 'Building personal tools by programming.' *Communications of the ACM*, 43 (8), 92-95.

SMITH, G. & GRANT, B. 2000. 'From players to programmers: a computer game design class for middle-school children.' *Journal of Educational Technology Systems*, 28 (3), 263-275.

SMITH, G. & SULLIVAN, A. 2012. 'The five year evolution of a game programming course.' In: SMITH KING, L., MUSICANT, D., CAMP, T. & TYMANN, P. (eds.) *Proceedings of the 43rd Technical Symposium on Computer Science Education*. Raleigh, USA, 29 February-3 March. New York, USA: ACM. 87-92.

SOLOWAY, E. & SPOHRER, J. (eds.) 1989. *Studying the novice programmer.* Hillsdale, USA: Lawrence Erlbaum Associates.

SPOHRER, J. & SOLOWAY, E. 1989. 'Novice mistakes: are the folk wisdoms correct?' In: SOLOWAY, E. & SPOHRER, J. (eds.) *Studying the novice programmer*. Hillsdale, USA: Lawrence Erlbaum Associates. 401-416.

STAHL, G., KOSCHMANN, T. & SUTHERS, D. 2006. 'Computer-supported collaborative learning.' In SAWYER, R. (ed.) *The Cambridge handbook of the learning sciences*. New York, USA: Cambridge University Press. 409-425.

SQUIRE, K. 2004. *Replaying history: learning world history through playing Civilization III.* PhD thesis, University of Indiana, USA.

SQUIRE, K. 2005. 'Changing the game: what happens when video games enter the classroom?' [Online]. Available: http://www.editlib.org/p/107270/ [Accessed 16/01/15].

STAGER, G. 2007. *An investigation of constructionism in the Maine Youth Center.* PhD thesis, University of Melbourne, Australia.

STAGER, G. 2008. 'A new paradigm for evaluating the learning potential of an ed tech activity.' *Proceedings of the Australian Computers in Education Conference*. Canberra, Australia, 29 September-2 October. Lesmurdie, Australia: ACCE. 467-479.

STEWART-GARDINER, C., CARMICHAEL, G., LATHAM, J., LOZANO, N. & GREENE, J. 2013. 'Influencing middle school girls to study computer science through educational computer games.' *Journal of Computing Sciences in Colleges*, 28 (6), 90-97.

STEVENSON, D. 1997. *Information and Communications Technology in UK schools: an independent inquiry*. Independent ICT in Schools Commission.

STILLER, E. 2009. 'Teaching programming using bricolage.' *Journal of Computing Sciences in Colleges*, 24 (6), 35-42.

STOKES, K. 2014. 'Who are the UK's young digital makers?' [Online] Available: http://www.nesta.org.uk/blog/who-are-uk%E2%80%99s-young-digital-makers. [Accessed 23/06/14].

STOLEE, K. & FRISTOE, T. 2011. 'Expressing computer science concepts through Kodu Game Lab.' In: CORTINA, T., LOWENFELD WALKER, E., SMITH KING, L. & MUSICANT, D. (eds.) *Proceedings of the 42nd Technical Symposium on Computer Science Education*. Dallas, USA, 9-12 March. New York, USA: ACM. 99-104.

SWACHA, J., SKRZYSZEWSKI, A. & SYSLO, W. 2010. 'Computer game design classes: the students' and professionals' perspectives.' *Informatics in Education*, 9 (2), 249-60.

SWELLER, J. 1994. 'Cognitive load theory, learning difficulty, and instructional design.' *Learning and Instruction*, 4 (4), 295-312.

TEAGUE, D. 2014. ‘Neo-Piagetian theory and the novice programmer.’ In: DU BOULAY, B. & GOOD, J. (eds.) *Proceedings of the 25th Psychology of Programming Annual Conference.* Brighton, England, 25-27 June. Brighton: University of Sussex. 203-206.

TES 2014. *ICT and Computing forum* [Online]. Available: http://community.tes.co.uk/tes_ict_and_computing/f/22.aspx [Accessed 07/09/14].

TESK, P. & FRISTOE, T. 2010. ‘‘Let the players play!’ & other earnest remarks about videogame authorship.’ In: GOMEZ, K., LYONS, L. & RADINSKY, J. (eds.) *Proceedings of the 9th International Conference of the Learning Sciences.* Chicago, 29 June-2 July. Chicago, USA: ISLS. 166-173.

TESLER, L., SMITH, D. & CYPHER, A. 1997. *Stagecast Creator*. [Computer program]. Stagecast.

THE LEAD PROJECT 2012. *Super Scratch programming adventure*. San Fransisco, USA: No Starch Press.

THOMAS, P. & MARTIN, E. 2008. ‘Using a phenomenographic approach in evaluating hypermedia stories.’ *Computers & Education*, 50 (2), 613-626.

THOMPSON, E. 2007. ‘Holistic assessment criteria - applying SOLO to programming projects.’ In: MANN, S. & SIMON (eds.) *Proceedings of the 9th Australasian Computing Education Conference*. Victoria, Australia, 30 January-2 February. Darlinghurst, Australia: Australian Computer Society. 155-162.

THOMPSON, E., LUXTON-REILLY, A., WHALLEY, J., HU, M. & ROBBINS, P. 2008. ‘Bloom's taxonomy for computer science assessment.’ In: HAMILTON, S. & HAMILTON, M. (eds.) *Proceedings of the 10th Australasian Computing Education Conference*. Wollongong, Australia, 22-25 January. Darlinghurst, Australia: Australian Computer Society. 155-161.

TIONG, K. & YONG, S. 2008. ‘Learning through computer game design: possible success (or failure) factors.’ In: CHAN, T., BISWAS, G., CHEN, F., CHEN, S., CHOU, C., JACOBSON M., KINSHUK, R. et al. (eds.) *Proceedings of the 16th International Conference on Computers in Education*. Tapei, Taiwan, 27-31 October. Jhongli City, Taiwan: Asia-Pacific Society for Computers in Education. 947- 951.

TOBIAS, S. & FLETCHER, J. (eds.) 2011. *Computer games and instruction*. Charlotte, USA: Information Age Publishing Inc.

TUCKER, A. 2006. *A model curriculum for K-12 computer science: final report of the ACM K-12 task force curriculum committee*. 2nd edition. CSTA.

TURKLE, S. 2003. ‘From powerful ideas to PowerPoint.’ *Convergence: The Journal of Research into New Media Technologies*, 9 (2), 19-25.

TURKLE, S. & PAPERT, S. 1990. ‘Epistemological pluralism: styles and voices within the computer culture.’ *Signs*,16 (1), 128-157.

ULICSAK, M. & WILLIAMSON, B. 2010. *Computer games and learning*. Futurelab.

VICTOR, B. 2012. ‘Learnable programming.’ [Online]. Available: http://worrydream.com/LearnableProgramming/ [Accessed 29/09/12].

VOS, N., VAN DER MEIJDEN, H. & DENESSEN, E. 2011. 'Effects of constructing versus playing an educational game on student motivation and deep learning strategy use.' *Computers & Education*, 56 (1), 127-137.

VYGOTSKY, L. 1978. *Mind in society: the development of higher psychological processes*. Cambridge, USA: Harvard University Press.

WALLER, D. 2009. *Basic projects: Game Maker*. Oxford: Payne-Gallway.

WEBB, M. 2002. Pedagogical reasoning: issues and solutions for the teaching and learning of ICT in secondary schools. *Education and Information Technologies*, 7 (3), 237-255.

WEBB, M. & COX, M. 2007. *Learning ICT inside the black box*. London: NFER Nelson.

WEGERIF, R. & DAWES, L. 2004. *Thinking and learning with ICT*. London: Routledge Falmer.

WENGER, E. 1999. *Communities of practice: learning, meaning, and identity.* Cambridge: Cambridge University Press.

WERNER, L., CAMPE, S. & DENNER, J. 2012a. 'Children learning computer science concepts via Alice game-programming.' In: SMITH KING, L., MUSICANT, D., CAMP, T. & TYMANN, P. (eds.) *Proceedings of the 43rd Technical Symposium on Computer Science Education*. Raleigh, USA, 29 February-3 March. New York, USA: ACM. 427-432.

WERNER, L., DENNER, J., BLIESNER, M. & REX, P. 2009. 'Can middle-schoolers use Storytelling Alice to make games? Results of a pilot study.' In: WHITEHEAD, J. & YOUNG, M. (eds.) *Proceedings of the 4th International Conference on the Foundations of Digital Games*. Florida, USA, 26-30 April. New York, USA: ACM. 207-214.

WERNER, L., DENNER, J. & CAMPE, S. 2012b. 'The fairy performance assessment: measuring computational thinking in middle school.' In: SMITH KING, L., MUSICANT, D., CAMP, T. & TYMANN, P. (eds.) *Proceedings of the 43rd Technical Symposium on Computer Science Education*. Raleigh, USA, 29 February-3 March. New York, USA: ACM. 215-220.

WHITEHEAD, J. 2008. 'Introduction to game design in the large classroom.' In: YOUNG, M. (ed.) *Proceedings of the 3rd International Conference on Game Development in Computer Science Education*. Miami, USA, 28 February-3 March. New York, USA: ACM. 61-65.

WHITEMAN, M. 2008. 'SOLO Taxonomy.' [Online]. Available: http://www.slideshare.net/mikeict/solo-taxonomy-484849 [Accessed 18/01/13].

WILLETT, R. 2005. 'New models of learning for new media: observations of young people learning digital design.' In: BACHMAIR, B., DIEPOLD, P. & DE WITT, C. (eds.) *Jahrbuch medienpädagogik*, *4*. Wiesbaden, Germany: VS Verlag für Sozialwissenschaften. 127-144.

WILLETT, R. 2007. 'Technology, pedagogy and digital production: a case study of children learning new media skills.' *Learning, Media and Technology*, 32 (2), 167-181.

WILLIAMSON, B. 2009. *Computer games, schools and young people: a report for educators on using games for learning*. Futurelab.

WILSON, A., HAINEY, T. & CONNOLLY, T. 2012. 'Evaluation of computer games developed by primary school children to gauge understanding of programming concepts.' In FELICIA, P. (ed.) *Proceedings of the 6th European Conference on Games Based Learning*. Cork, Ireland, 4-5 October. Reading: Academic Publishing International Ltd. 549-558.

WING, J. 2006. 'Computational thinking.' *Communications of the ACM*, 49 (3), 33-35.

WING, J. 2008. 'Computational thinking and thinking about computing.' [Online]. Available: http://rsta.royalsocietypublishing.org/content/366/1881/3717 [Accessed 10/01/15].

WJEC 2012. *GCSE Computer Science specification*. WJEC.

WOOLLARD, J. (ed.) 2009. *Computer programming in Key Stage 3*. CAS.

YANG, Y. & CHANG, C. 2013. 'Empowering students through digital game authorship: enhancing concentration, critical thinking and academic achievement.' *Computers & Education*, 68, 334-344.

YAROSLAVSKI, D. 2008. *Lightbot*. [Computer game]. Armor Games.

YATIM, M. & MASUCH, M. 2007. 'Gatelock - a game authoring tool for children.' In: SKOV, M. (ed.) *Proceedings of the 6th International Conference on Interaction Design and Children*. Aalborg, Denmark, 6-8 June. New York, USA: ACM. 173-174.

YEH, K. 2009. 'Using an educational computer game as a motivational tool for supplemental instruction delivery for novice programmers in learning computer programming.' In: GIBSON, I., WEBER, R., MCFERRIN, K., CARLSEN, R. & WILLIS, D. (eds.) *Proceedings of the Society for Information Technology & Teacher Education International Conference*. Charleston, USA, 2-6 March. Chesapeake, USA: AACE. 1611-1616.

YIN, R. 2009. *Case study research: design and methods*. Thousand Oaks, USA: Sage.

YOYO GAMES 2007. *Game Maker 7.* [Computer program]. YoYo Games Ltd.

YOYO GAMES 2014. 'Learn how to use Game Maker: Studio' [Online]. Available: http://yoyogames.com/learn [Accessed 10/05/14].

ZAGAMI, J. 2008. 'Which programming language makes it easier for students to learn to program?' *Paper presented at the Australian Council for Computers in Education Conference*, Canberra, Australia, 29 September-2 October.

ZIMMERMAN, E. 2009. 'Gaming literacy: game design as a model for literacy in the twenty-first century.' In: PERRON, B. & WOLF, M. (eds.) *The video game theory reader 2*. New York, USA: Routledge. 23-31.

ZORN, I. 2008. 'Active construction of digital media as socio-technical construction of a learning space.' In: LUCA, J. (ed.) *Proceedings of the World Conference on Educational Multimedia, Hypermedia and Telecommunications*. Vienna, Austria, 30 June. Chesapeake, USA: AACE. 4534-4543.

ZORN, I. 2009. ‘Educational construction of information technology as engagement with the course of the world.’ In: BAMMÉ, A., GETZINGER, G. & WIESER, B. (eds.) *Yearbook 2008 of the Institute for Advanced Studies on Science, Technology and Society*. Munich, Germany: Profil. 341-366.

Appendices

Appendix 1: Analysis of pupil authored games

In this appendix the games pupils created are described to give an overall picture of what was achieved in terms of the game design and programming concepts evidenced in them and to provide a summary of the main difficulties encountered. These evaluations are then represented as matrices which show the relative strengths of each game, and the qualitative variation between games, using the SOLO taxonomy levels described in Chapter 4 (see Table 3).

AEMD
KS2 SAT average 5.67/5.66; CAT average 119; Jesson band high/high.

Intro screen, title screen, and 3 levels of 'Kokoro'.

This game ranks 1st out of 12 with a score of 58/80. It is the most fully-featured game of the group and the only game to contain 3 levels where the gameplay on each level is different. These boys worked confidently and independently and made use of sample games and printed tutorials. They were very engaged in the activity and worked on elements of the games at home and shared items via email.

Code organisation and documentation
This pair completed 11/12 planning documents and planned their game in great detail. Most game assets are correctly named and stored in folders to effectively manage game resources.

Problems
The main problems in their game are to do with 'incompleteness'. Although score, health and lives are fully implemented in level 1, they are absent from the other two levels. There is no background sound in levels 2 and 3 and no discernible way to progress from level 2 to 3. In level 2, some vertically scrolling cars appear to lie on top of each other, because they were programmed to return to view randomly across the width of the screen; in level 3 doors do not function as barriers to the player character as intended and when all reward objects are collected the win state does not correctly implement. On this level also the movement of the player character is not accurate.

Game storyline
In *Kokoro*, Takeshi (an archaeologist who possesses an ancient and valuable scroll) is captured by adversaries. His daughter, Kokoro, negotiates three game levels to release him.

Usability
These boys enjoyed customising their game and added features such as an animated intro screen, a loading bar, a game icon and a title screen. No instructions to play the game are given, but options to escape, save and load the game are implemented. Common controls are used (arrow keys for directional movement; space bar to fire missiles and to start the game). Messages are used to communicate with the player when a level has been completed. The pair created animated sprites for level 1; on level 3 sprites change direction left, right, up and down to make object movement realistic.

Functionality
The game features a splash screen, a title screen and 3 levels. A progress graphic displays while the game loads and a customised game icon appears in the title bar. An animated intro screen displays the game credits; the title screen appears. User options on the title screen function correctly. Level 1 is fully functional; the player controls the movement of the player plane by using the arrow keys and can fire missiles at enemy planes which explode and disappear when hit. Enemy bullets are fired in the direction of the player's plane; health and lives are lost when it is hit. On level 2, the player controls a car left, right, forwards and backwards using the arrow keys, to avoid colliding with oncoming cars. If there is a collision, an explosion animation plays, the player's car disappears and all instances of the other car are destroyed (in error). Scrolling backgrounds function correctly on levels 1 and 2. On level 3 the player moves a character around a maze, collecting keys to gain points. When the keys are collected they disappear; if the player character collides with an enemy object they reappear and she is returned to the start. This level is incomplete and the win state does not implement.

Scoring
Score, health, and lives mechanics are functional and displayed as on-screen graphics on level 1, but not in levels 2 or 3. A high score table displays when all lives are lost on level 1. A win/lose state is implemented for level 1.

Gameplay
The player can progress from level 1 to 2, but not beyond. The goal of the game is to free a captured character on level 3. On level 1 the player controls an aeroplane which flies over an ocean, firing missiles at enemy planes and avoiding enemy fire. The player has 3 minutes to gain points without losing lives/health. On level 2 the player controls a car as it drives across town and avoids oncoming vehicles. On level 3 the player controls Kokoro as she negotiates a maze to free her father from captivity, although the win state on this level cannot be reached due to programming errors. Game challenge is set too high in level 1 and level 3 and too low in level 2.

Sound
Background sound accompanies the intro screen and level 1. Sound effects indicate missiles have been fired or the player has been hit on level 1 and objects have been collected on level 3.

Game design
Level 1 and level 2 are vertically scrolling shooters - 'top down' view; level 3 is a maze. The player character object differs on each level (plane, car, girl), as do the non-player characters (planes, cars, henchmen). Collectable objects feature on level 3. Obstacles are present in all levels (enemy planes, cars, bullets, henchmen). There is a narrative underlying the game which supports coherence, although settings for each level differ (sea, road, maze). A Japanese theme is apparent in the game's title, title screen graphics and background sound. The player character is a girl, but she only appears in level 3 – in levels one and two she is 'implied'.

Programming
This pair understood and applied the concept of event-driven programming and used 84 events (*step, alarm, collision, create, outside room, animation end, keyboard, no more lives, no more health, draw*) and 170 actions to create the game play. Thirty-one conditional statements were used to test variables: *check sound*, *test instance count*, *test chance*, *check grid*. Six *alarm* events and 15 *step* events were used to create loops in the game. Nine different variables were used to store data in the game (lives, health, score, vspeed, speed, x/y); variables were tested (room height/width); one variable was created. The logical operator NOT and relational operators <, >, /, = were used in expressions. Boolean logic (true/false) is implied in conditionals and used to play looping sound, to redraw the screen and to define objects as solid. Coordinates were used to indicate position, to draw lives and to draw health. Negative number was used to define health, lives and coordinates. Randomness was used to define object position. Relative values were used to define health, score, lives, position.

Graphics
This pair created an animated splash screen and animated sprites, loading bar, title screen and game icon to customise their game. Some graphics were sourced from *Game Maker* resources. Others were created in *Game Maker's* sprite editor and *Fireworks*.

The matrix below maps the SOLO score achieved for each component:

Extended	10								
	9								
Relational	8								
	7								
Multistructural	6								
	5								
Unistructural	4								
	3								
Pre-structural	2								
	1								
		Usability	**Functionality**	**Scoring**	**Game play**	**Sound**	**Overall design**	**Programming**	**Graphics**

ACJC
KS2 SAT average 5.49/5.03; CAT average 122/121; Jesson band high/above average.

Title screen and 3 levels of 'Ultimate Dodgeball'

This game ranks 2nd out of 12 with an overall score of 49/80. These boys were very engaged in the game authoring activity and worked independently throughout, although they were frustrated that they had to change their initial game ideas to avoid negative representations of certain groups.

Problems
They also expressed frustration with the lack of advanced tuition in the video tutorials. They had some difficulty establishing a storyline for their game, with controlling the movement of the player character and with displaying lives on the screen. They also encountered problems with their scoring system and advancing through levels. Much of their learning talk involves finding, creating and editing graphics. In fact, this pair found sourcing, creating, and editing graphics frustrating and time consuming and thought that more graphics should be made available in the software; they preferred to spend their time on creating the game play.

Code organisation and documentation
Most game assets are named correctly and there are few extraneous items. 2/3rds of the planning documents were completed. Absence and the need to merge games caused some problems.

Game storyline
In *Ultimate Dodgeball*, a boy must avoid dodgeballs in a dodgeball arena and collect Mars bars to gain points.

Usability
A title screen adds realism to the game but there are no game instructions. Common control keys are used; score and lives are displayed on screen as text. Messages and a high score table give feedback to the player. Player progression is achieved over three levels.

Functionality
The game functions adequately as a playable game and is reasonably complete, although there are limited interactions. The player character moves as intended and score and lives function correctly. Movement between levels is achieved by gaining a certain number of points; messages congratulate the player if a level is completed and commiserate when all lives are lost; a high score table ranks scores.

Scoring
The score works correctly - starting at 0 and increasing to a set amount for each level. Lives display on screen and decrease correctly, although sometimes register as -1. The high score table displays correctly. Win state: if 980 points are collected on level 3, the player is congratulated and the high score table appears.

Gameplay
The player starts with 3 lives and uses the arrow keys to control the player character, who gains points by catching Mars bars and avoiding dodgeballs. Catching Mars bars adds 20 points to the player's score; colliding with a dodgeball loses a player life. The purpose of the game is to gain as many points as possible and advance through the levels until the final level is reached and the high score table displays. Progression through the levels is achieved by scoring a certain number of points on each level. Level play is differentiated by increasingly fewer Mars bars to catch and obstacles (balls) falling at increased speed. Challenge doesn't significantly vary between levels and there are limited interactions between the player and the game, and within the game itself.

Sound
No background sound plays, because the wrong sound file was selected in the *play sound* action used in the *create* event of the player character. No sound effects are implemented.

Game design
'Ultimate Dodgeball' is an action game set in a notional dodgeball arena. The game features a player character (boy, cartoon character), reward object (Mars bar) and obstacle (balls). The game is structured coherently overall in so far as it functions as a playable game over 3 levels. Gameplay is the same on each level and features the same player character and reward/obstacle objects.

Programming
These pupils understood the concept of events and used them effectively as inputs. Twenty-four events and 50 actions are used to create the game play. This pair used a wider range of actions than any other pair. Conditional statements were correctly implemented to test lives and score. Variables are used to store data in the game. Understanding of Boolean logic is implied in the use of conditional statements and also used to set object properties to 'solid' and 'looping sound' to 'true'. Several mathematical concepts were also used as necessary (relational operators (=), coordinates to specify position, negative number, randomness and relative value).

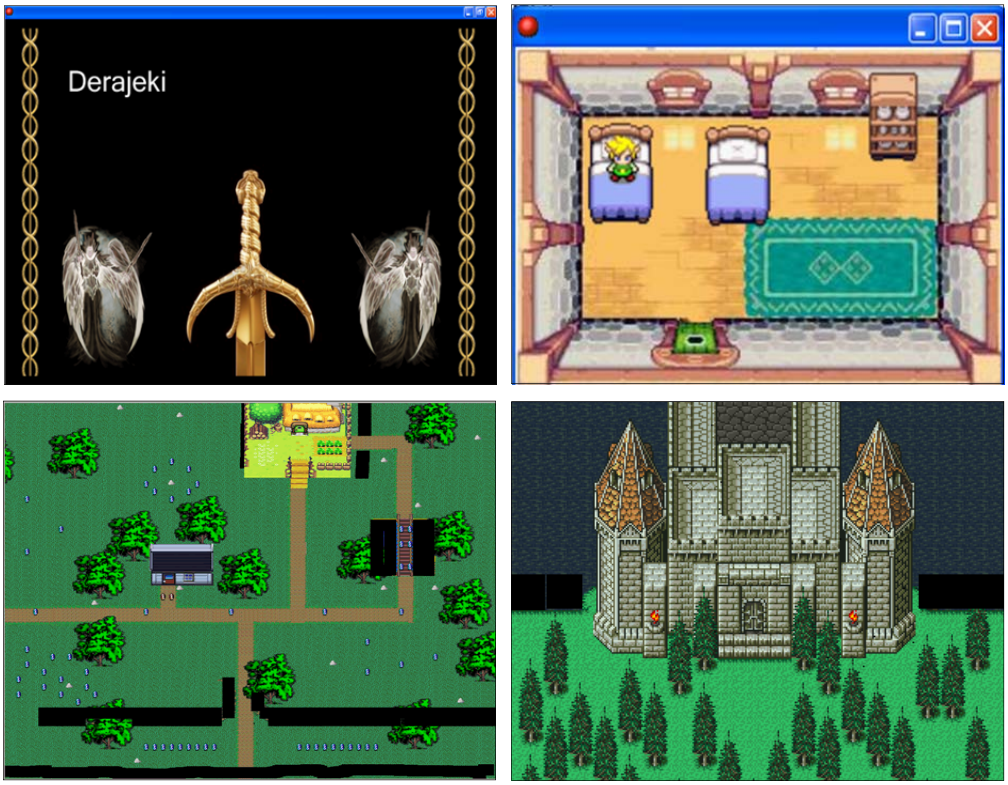
Graphics

Game graphics were sourced from the internet and modified using *Fireworks*. The game backgrounds are rudimentary, and where the game space could have been a representation of a dodgeball arena, instead, wall sprites sourced from *Game Maker* resources were used as the background in level 1. No backgrounds were loaded for levels 2 and 3; wall objects bound the playable space on each level and change colour to differentiate the levels. The title screen was created in *Fireworks*, as was the 'Start' button. Its title incorporates a dodgeball as the letter 'o' of 'dodgeball' and features a logo for dodgeball.com (a defunct mobile social networking service). The graphics are a mix of photos, vector graphics, line drawings and there is no unifying theme or colour scheme, but these pupils used professional graphics editing software effectively to resize and modify images to suit their purpose.

The matrix below maps the SOLO score achieved for each component:

Extended	10								
	9								
Relational	8								
	7								
Multi-structural	6								
	5								
Uni-structural	4								
	3								
Pre-structural	2								
	1								
SOLO level		**Usability**	**Functionality**	**Scoring**	**Game play**	**Sound**	**Overall design**	**Programming**	**Graphics**

JBLA
KS2 SAT average 4.67/4.51; CAT average 98/94; Jesson band average/average



JBLA Title screen, level 1, outside view

This game ranks 3rd out of 12 with an overall score of 40/80.

Code organisation and documentation
This pair partially completed only 3/12 of the planning documents and departed from their initial ideas. Naming of sprites, objects and rooms is not efficient.

Problems
The main problem for this pair was in implementing the score, which does not function correctly because the score is set in error in the *create* event of the collectable items and increases to 730 as soon as the player character enters the outside view. The player character travels off the screen to the right and left, and at the top, and disappears behind other objects in the outside view. A black screen displays for 3 seconds when the player character enters the outside view and there are some problems with the timing and sequence of animations and messages.

Game storyline
In *Derajeki* a boy explores a house and must avoid setting an 'evil dude' free. To gain points he must exit the house into an outside view where he collects rupees. This pair tried to recreate a *Legend of Zelda* (Nintendo) game and feature the series' protagonist, Link, as their player character.

Usability
A title screen displays a message to tell the player how to start the game but there are no other game instructions. Credits are written but not implemented. Messages are used to display instructions and to communicate with the player (Click the sword; It's locked; Help me; Please help; If you touch me you get a wish; Ha ha you fool, I'm

free!). The animated player character is controlled using the arrow keys and changes appearance on direction change. A script sourced from the internet is used to achieve realistic 'running' movement. Three other animated sprites are used (explosion, rupees, evil dude). Backgrounds are also sourced from the internet. Levels are linked thematically (5 room interiors and one outside view).

Functionality
On game start a title screen appears. The player clicks a graphic of a sword to move to level 1. The player character can move between rooms in Link's house and along a path in the outside view. Rupees disappear when collected but no score is added. The game is not implemented sufficiently to have a clear goal.

Scoring
The score is set in the *create* event of the collectable items (blue and red rupees) rather than in a *collision* event between Link and the rupee object, so does not function as intended, and no points are gained for collecting rupees. There is no high score table or win state.

Gameplay
On the title screen a message displays - 'Click the sword :)'. The first game room appears and music plays, looping (although there is a gap between loops). The player controls Link as he explores different rooms (room 1, a bedroom, has one door which leads to room 2, a kitchen. Here, there are 3 doors - one which leads back to the bedroom, another which leads to a furnace room and a third which leads outside). The player can move around in and out of rooms and along a pathway in the outside view. When the player character enters the outside view the game room enlarges and Link can explore the terrain and collect rupees to gain points. Game play is interrupted when Link inadvertently disappears behind part of the background graphic. If he re-enters the cottage he is returned to the kitchen. If he enters a second building he is faced with 2 doors. One doesn't open. The other generates a message saying 'It's locked'. A staircase leads to another room upstairs. Here, a message appears - 'Help me' and 'If you touch me you get a wish'. If the player clicks on this message there is an explosion and an animated 'evil dude' figure appears. When the player clicks this creature, a message displays - 'Ha ha you fool, now I am free', and returns Link to the start of the game.

Sound
Background music plays on game start. A 60 second pause follows before the music restarts. A 'chime' sound plays when the outside view is entered, instead of when instances of rupees are collected, as intended.

Game design
This adventure game is presented in 'top down' view and incorporates two locations: inside a cottage, and outside. The player controls an animated character as he explores his environment. The game view follows the movement of the player character.

Programming
This pair understood the concept of event-driven programming and used 45 events (*create, step, end step, collision, mouse, game start, animation end*) and 81 actions to create the game play. 31/45 events were collisions. They used one conditional statement to check 'if next room exists'. One *step* event was used to execute the player character's movement script in a continuous loop. Variables were used to store data in the game - to set speed, set variable (image speed), set variable (depth), set health, set score. Boolean logic was used to define whether a sound file should loop and to

define an object as solid. Coordinates were used to indicate position. Negative number was used to define depth, and health. Relative value was applied to health and score. This pair sourced a GML script online and interpreted code comments correctly to implement the script. They also learned how to insert a script action in *Game Maker*.

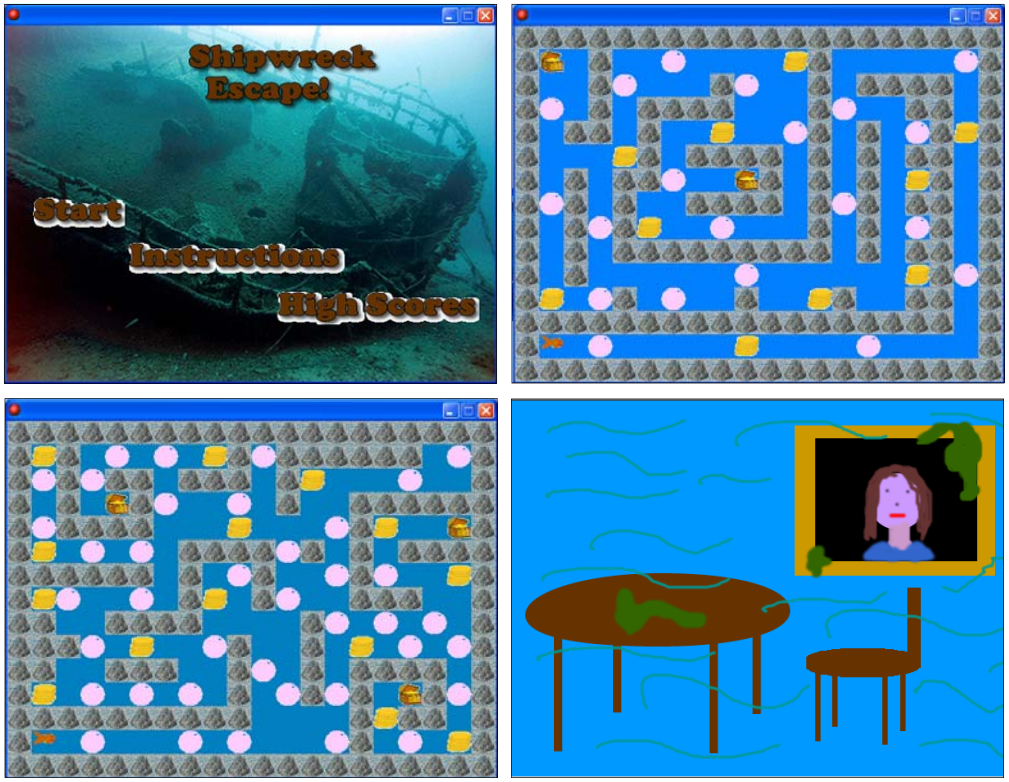
Graphics

Their game backgrounds use a mix of graphics from 'The Legend of Zelda: Minish Cap' (Nintendo) sourced online, and others they created themselves. Their title 'Derajeki' and the title screen are not linked thematically with other game graphics, either visually or narratively, although a sword features in the Minish Cap backstory. The animated player character was sourced online. There is some unevenness in the 'look and feel' of the graphics, and between the inside and outside view.

The matrix below maps the SOLO score achieved for each component:

Extended	10								
	9								
Relational	8								
	7								
Multistructural	6								
	5								
Uni-structural	4								
	3								
Pre-structural	2								
	1								
		Usability	**Functionality**	**Scoring**	**Gameplay**	**Sound**	**Overall design**	**Programming**	**Graphics**

KW
KS2 SAT average 4.76; CAT average 99; Jesson band average.



KW title screen, level 1, level 2

This girl chose to work alone. Her game ranks 4th equal with a score of 34/80. She worked independently, but thought the project was 'kind of a long process'.

Code organisation and documentation
KW completed 7/12 planning documents. She named 12/17 sprites/objects using spr/obj as a suffix instead of a prefix. Background and rooms have no prefix. There is 1 redundant room and 1 unnamed object (start button).

Problems
The main problem for this pupil was level progression. The *test score* action was applied to the wrong event so did not function. There is no challenge in the game because no enemy characters were implemented and there is no win/lose state. She had difficulty in locating a suitable enemy character sprite (shark) and in importing graphic resources.

Game storyline
In *Shipwreck Escape* (a maze game), Patrick the fish has to negotiate a maze of rocks and collect pearls, coins and treasure chests to gain points, while avoiding crabs.

Usability
Game instructions were attempted but incorrectly implemented as an *execute code* action which caused a fatal error message to appear - neither game nor instructions can be accessed. Arrow keys are used for directional movement. The score displays on screen. There is no animation and the player character does not change sprite on direction change. There are two thematically linked levels but no mechanism to pass to level 2.

Functionality
The title screen offers user options to start the game and to view a high score table. The player character moves correctly in all directions on key press and stops on key release. The score implements correctly when reward objects are collected.

Scoring
The score mechanic functions correctly on increase but there is no mechanism to lose points. Lives are set but not correctly implemented to display on screen. No mechanism to lose lives is implemented. The scoring mechanism to progress to level 2 does not function. A high score table is partially implemented but the condition to display the high score table is incomplete.

Gameplay
The player controls a fish as he swims through a maze. The goal is to collect items of treasure to gain points and to reach a high score. Items disappear when they are collected. There is limited game play and challenge because the enemy character/obstacles were not implemented so there is no mechanism for losing lives or points. There is no level progression due to a programming error in the score mechanic. The level 2 maze seems easier to solve than level 1. Level 2 is incomplete (more enemies were planned for level 2 but not implemented).

Sound
No sound implemented.

Game design
The design theme is an underwater shipwreck. The object inventory is consistent with the theme: fish, shipwreck, treasure, (pearls, gold coins, and treasure chests). The game is structurally coherent. The same objects and interactions are used in both levels.

Programming
The game evidences that KW understood the concept of event-driven programming - she correctly used 17 events as input data (*create, collision, key press/release*) and 24 actions to create the game play. She used a conditional statement to test the score, but put this action in the *game end* event so it did not function correctly. Variables were used to store data in the game (*set score, set lives*); *test score* and *speed* were partially correctly implemented. Boolean logic is implied in the conditional statement and was used to define the maze wall object as solid. A relational operator (>) was used in the *test score* action. Relative value was used to increase score.

Graphics
This pupil created the player character (a crudely drawn fish) and the interaction buttons in *Fireworks*. Other sprites and backgrounds were sourced online or from *Game Maker* resources. She customised the high score table with a background image and red text.

The matrix below maps the SOLO score achieved for each component:

Extended	10								
	9								
Relational	8								
	7								
Multistructural	6								
	5								
Unistructural	4								
	3								
Pre-structural	2								
	1								
		Usability	**Functionality**	**Scoring**	**Game play**	**Sound**	**Overall design**	**Programming**	**Graphics**

OWSW
KS2 SAT average 5.5/N/A; CAT average 117/111; Jesson band high/above average.

OWSW Title screen, instructions, level 1

This game ranks 4th equal with a score of 34/80.

Code organisation and documentation
This pair completed 10/12 planning documents.
17/18 game assets are correctly named.

Problems
The main problems for this mixed gender pair lay in keeping objects in view on the screen in this vertically scrolling game. The collectable items and obstacles do not reappear when they have disappeared from view. Initially, when the player character collided with obstacles they became attached to it instead of disappearing, however this problem was later resolved. The score does not increase relative to the current value. The lives variable is not implemented correctly and does not display on screen.

Game storyline
In *Snow Mountain*, a snowboarder travels down a mountain, avoiding trees and rocks and collecting blue and yellow snowflakes to gain 10 and 20 points respectively. The player character loses lives and points if he collides with rocks or trees.

Usability
The title screen offers user options to start the game or to view the game instructions. Left and right arrow keys are used to control player character movement. The game is incomplete in so far as there is only one level, so there is limited gameplay/challenge and no level progression.

Functionality
The title screen buttons function correctly to start the game and to launch the game instructions. The game has some functionality. The player controls character movement using left and right arrow keys; movement stops when keys are released. There is partial score functionality. The background scrolls, but the obstacles and collectables do not reappear when they have scrolled off screen, so no interactions are possible after 10 seconds of game play.

Scoring
The score variable partially functions but does not increase relative to the current value, although it does decrease relatively. The score displays intermittently on screen. The life variable is incorrectly set at the start of the game and does not display on screen. No win/lose state is implemented.

Gameplay
The player controls the left/right movement of a snowboarder as he travels down a mountain to collect snowflakes for points. These items disappear when they are collected and a score action implements. The player loses points and lives if he collides with rocks and trees. The game room speed is set too fast and too few collectable items are placed in the room to enable the player to accumulate many points.

Sound
No sound implemented.

Game design
The game is structurally coherent - objects relate to the theme. Title screen and instructions screen are thematically linked to the main game and maintain the same colour scheme.

In their planning documents the intention was for the game to advance through 3 levels every successive 100 points gained; each level scrolls faster and more obstacles appear; the game ends after 1.5 minutes. If 3 lives are lost a high score table, a 'game over' message and a replay button should appear, but these features were not implemented.

Programming
This pair understood the concept of event-driven programming and used *mouse, keyboard, step, collision* and *create* events as input data. They used 18 events and 31 actions to create the game play. They attempted to use 5 conditional statements - the structure is correct for these but the arguments are not. A loop-like structure was attempted by using a *step* event to repeat an action - again the structure is correct but the arguments are not. Variables were used to store data for lives, score and speed, but errors in implementation mean that score and lives do not work correctly. Boolean logic was used to define objects as solid and implied in the use of conditionals. Relational operators were used in expressions (<, >, -) and coordinates were used to specify position. Negative number was used for lives, score, position and vspeed. Random values were used to indicate position and relative values were used for lives, position and score. This pair used an above average range of programming concepts in their game, although they were only partially correctly implemented.

Graphics
This pair used a mix of graphics they created themselves (tree, snowflakes, game background, interaction buttons, title) and others sourced online (title screen background, snowboarder, rock). There is no animation and the player character does

not change appearance on direction change. The title screen and instructions screens offer an attractive design but the interaction buttons are inconsistent.

The matrix below maps the SOLO score achieved for each component:

Extended	10								
	9								
Relational	8								
	7								
Multistructural	6								
	5								
Unistructural	4								
	3								
Pre-structural	2								
	1								
		Usability	**Functionality**	**Scoring**	**Game play**	**Sound**	**Overall design**	**Programming**	**Graphics**

JBJG
KS2 SAT average 4.92/4.5; CAT average 101; Jesson band above average/average

JBJG title screen, level 1, level 2

This game ranks 5th with a score of 33/80.

Code organisation and documentation
9/12 planning documents were completed. 12/19 game resources were correctly named. There is one redundant object and one redundant sprite.

Problems
The main problem for this pair is that they did not manage to display score or lives information on the screen. These variables were partially implemented (the score increases when carrots are collected but no carrots are placed in the room so the score cannot be seen; score and lives are not correctly set at the start of game; lives increase by 10 if the horse eats an apple - viewed in debug mode). They also had problems in making the collectable and obstacle items reappear after they had scrolled off screen; after several seconds of gameplay no further game interactions are possible. Initially they had problems in getting the player character, a horse, to move. Their intention was for the horse to jump but they did not manage to implement this. They had to simplify their ideas so that the horse's movement could be controlled by the arrow keys.

Game storyline
In *Shadey's Adventure*, a horse gallops through a forest at night, jumping over logs and gaining points and lives by eating carrots and apples.

Usability
The title screen offers a start button, but there are no game instructions. The player uses arrow keys to control the directional movement of an animated horse. Interface

design is consistent - the title screen links with the game levels but level 2 is not implemented. No score or lives status is visible.

Functionality
The game has some functionality - the player character can be moved in all directions and apples disappear when the horse eats them. Some apples do not appear in the correct position for the horse to be able to reach them, so limited interactions are possible. There is no functioning score mechanic. A lives mechanic is partially implemented but does not display on screen.

Scoring
The score mechanic is assigned to a carrot object, which is not placed in the game room so no score can be achieved or displayed. The score is not set to increase relative to the current value so remains at an absolute value.

Gameplay
The player can start the game and use the arrow keys to guide Shadey the horse through a forest. He can jump over logs and 'eat' apples. There are limited game interactions (collision with apple/increase lives) and no penalties. There is no mechanism to progress from level 1 to level 2. Level 2 is incomplete so there is no win/lose state. There is no challenge because obstacles and rewards are not fully implemented.

Sound
No sound implemented.

Game design
This horizontally scrolling game consists of a title screen and one level. There is one animated sprite. Game objects are thematically linked. The game is structurally coherent but level 2 is not implemented.

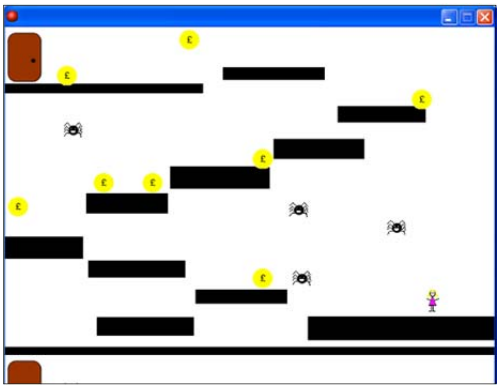
Programming
This pair showed some understanding of event-driven programming and used 12 events (*mouse, keyboard, create* and *collision*) and 19 actions to create the gameplay. They attempted to achieve a loop construct by using an *alarm* event but this was incomplete. They also tried to use variables to store data in the game (lives, score, speed, x/y) but these were only partially implemented so do not function as intended. 2/3 conditional statements were implemented correctly; the other (*test lives*) is incomplete. The *set score* action is incorrect - the score does not increase relative to the current value. The *draw lives* action is incorrect since it is not placed in a *draw* event; the *set lives* action is correct but lives do not accurately increment because the associated objects are not positioned correctly in the game room. Relational operators (>, <) are used in expressions. Coordinates are used to control the position of the horse and to indicate the screen location of the lives status. Negative number is used to define direction of movement and room speed. Relative value is applied to lives, and to the position of the horse.

Graphics
This pair used a mix of graphics they created themselves in *Fireworks* (apples, game background, title, start button, trees, logs) and others sourced online (animated horse, carrot). The background and trees are fairly well executed but the apple and carrot graphics do not have a transparent background as required.

The matrix below maps the SOLO score achieved for each component:

Extended	10								
	9								
Relational	8								
	7								
Multistructural	6								
	5								
Unistructural	4								
	3								
Pre-structural	2								
	1								
		Usability	**Functionality**	**Scoring**	**Game play**	**Sound**	**Overall design**	**Programming**	**Graphics**

CBMH
KS2 SAT average 5.33/5.42; CAT average 113/106; Jesson band high



CBMH title screen, level 1

This game ranks 6th with a score of 28/80.

Code organisation and documentation
8/12 planning documents were completed. 5/6 sprites are named without a spr_ prefix; prefixes are correctly used for 3/4 backgrounds and 1/2 rooms; a background graphic is loaded in error as a sprite; 1 sound file is loaded but there is no data in the file; duplicate castle backgrounds are loaded but not implemented.

Problems
This pair had problems with deciding on their game storyline and selecting an 'enemy' character. They did not have a clear vision for the game. The enemy objects (spiders) disappear after 10 seconds of game play. The player character travels off screen, although can be returned to view. The score displays intermittently. The game was intended to be a platform game but no jumping mechanic was implemented for the player character, whose movement is controlled by the arrow keys.

Game storyline
In *The Great Escape*, a princess is being chased by an evil witch through a castle, collecting coins and avoiding spiders.

Usability
The title screen offers a start button but there are no game instructions. Common control keys are used for directional movement of the player character. The score increases correctly when coins are collected, but disappears from view if the player character collides with a spider. There is no animation.

Functionality
The start button functions correctly. The player can control the movement of the player character using the arrow keys and can score points. Coins disappear when collected. When the player character collides with a spider, both objects stop moving but no other actions are implemented for this event.

Scoring
Points are scored when coins are collected; the score increases relative to its current value. Points are lost when the player character collides with a spider, but not relative to the current value. Lives are lost when the player character collides with a spider but the lives status does not display on screen.

Gameplay
The player can move the main character and collect coins to gain points. The player loses a life if s/he collides with a spider. There are limited interactions in this single level game and there is no win/lose state or level progression.

Sound
No sound implemented.

Game design
The coherence of this game suffers because the girls changed their ideas and did not manage to implement all aspects of their initial design.

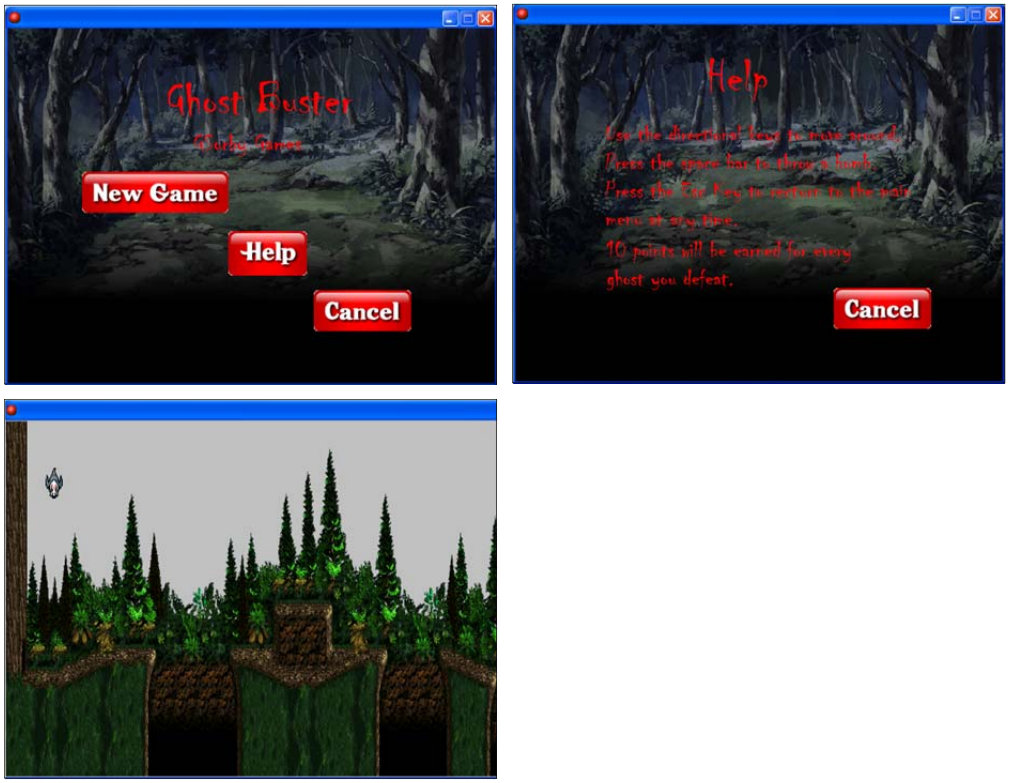
Programming
This pair showed some understanding of event-driven programming and applied 8 events (*create, collision, key press*) and 12 actions to create the game play. They attempted to use a conditional statement but did not manage to implement it correctly. Variables are used to store data in the game (*set score*, *speed* and *gravity*) but are not always fully developed or correctly implemented. No values are entered for coordinates and angles in the two actions which require them. Negative number is applied to decrease score. Relative value is applied correctly to increase, but not to decrease the score.

Graphics
This pair created the graphics used in their game using *Fireworks* (girl, spider, coin, platform, door, start button, title), but these are very simply executed. The girl's head is transparent so cannot be seen when she passes in front of the black 'platforms'. The dimensions of the player character are too small.

The matrix below maps the SOLO score achieved for each component:

Extended	10								
	9								
Relational	8								
	7								
Multistructural	6								
	5								
Unistructural	4								
	3								
Pre-structural	2								
	1								
		Usability	**Functionality**	**Scoring**	**Game play**	**Sound**	**Overall design**	**Programming**	**Graphics**

GS
KS2 SAT average; CAT average; Jesson band - no data available



GS title screen, help screen, level 1

This pupil worked alone. His game ranks 7th with a score of 27/80.

Code organisation and documentation
This boy completed 3/12 planning documents. Game assets are not correctly named with prefixes. There is one unnamed object. Several sprites, sounds, backgrounds and objects, are loaded into the game resources but are not implemented in the game.

Problems
The main problem in this game is that there are limited interactions. The game is not 'playable'. This pupil spent much of his time locating graphics and sound files and creating a title screen and interaction buttons with rollover effects.

Game storyline
Ghost Buster is set in a forest and features a player character and a ghost, but it is difficult to discern a storyline from the pupil's planning documents. The player character has to 'defeat various enemies and collect items to enhance power and speed' and must defeat a boss character 'who attacks with a Ball of Darkness'.

Usability
The title screen features a title, credits and three buttons (New Game, Help, Cancel). The buttons change appearance on rollover. Brief game instructions are given although the instructions for the Escape button are not correct (the Escape button is intended to return the user to the main menu but actually quits the game). Arrow keys are used to control the directional movement of the player character. Interface design is consistent across title screen and level 1. The player character changes appearance on direction change (left and right). One level offers limited user interaction.

Functionality
When the game is run, music plays and the title screen appears. All interaction buttons on the title screen work correctly. The 'New game' button launches level 1, where the player can achieve effective movement of the player character with the arrow keys.

Scoring
No score implemented.

Gameplay
The player can navigate between the title and help screens, launch or exit the game and control the directional movement of the player character. No other interactions are implemented. No score mechanic is implemented. There is no gameplay.

Sound
Looping background sound plays on game start. Three sound files are loaded in the game resources but only one is programmed to play.

Game design
The game is incomplete and consists of a title screen, a help screen and one level.
All screens are narratively/visually coherent but only one object is implemented.

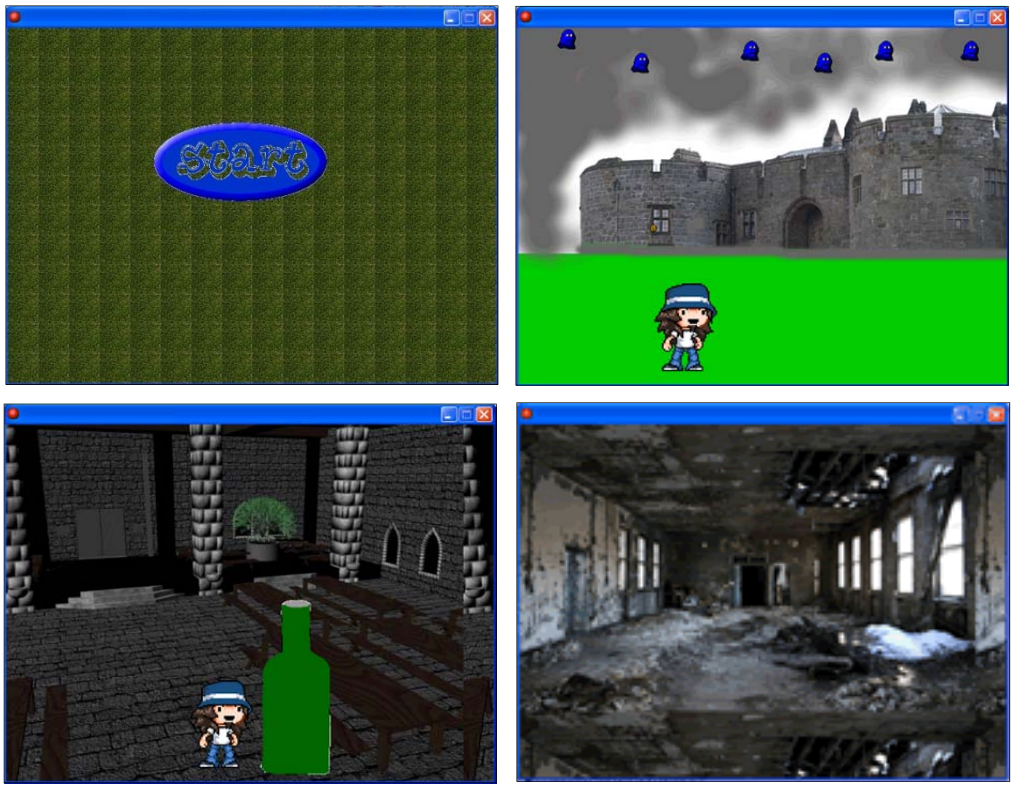
Programming
The game evidences that this boy understood and applied the concept of event-driven programming, although most events control game navigation. *Key press* and *release* events are correctly used as input data to control object movement. Twenty-five events and 28 actions are used to create user interaction. No conditional statements are used and there is limited use of variables - a speed variable is set to initiate and to stop movement. Boolean logic is applied to loop the playback of the sound file.

Graphics
This pupil created the interaction buttons and title in *Fireworks*. Other graphics (ghost and background images) were sourced online. The buttons are of consistent design and change appearance on rollover, although the button canvas is not transparent. Some sprites and objects created are not implemented.

The matrix below maps the SOLO score achieved for each component:

Extended	10								
	9								
Relational	8								
	7								
Multistructural	6								
	5								
Unistructural	4								
	3								
Pre-structural	2								
	1								
		Usability	**Functionality**	**Scoring**	**Game play**	**Sound**	**Overall design**	**Programming**	**Graphics**

JDMB
KS2 SAT average 4.86/5.05; CAT average 100/114; Jesson band above average



JDMB title screen, level 1, level 2, level 3

This game ranks 8th with a score of 26/80.

Code organisation and documentation
4/12 planning documents were completed. Sprites are unnamed. Objects are named without a prefix. 1 background is duplicated.

Problems
These boys found it frustrating that they had to modify their initial game ideas to avoid depictions of violence. They found it difficult to translate their ideas into game action. The main problem within the game is the control of the player character's movement. He moves left and right but does not stop on key release and travels off the screen. On level 2, the movement of the player character is tied to the movement of the brick object, in error.

Game storyline
In *Zombie Nation*, the last man on earth must defend a castle from zombie attack and must destroy these enemies by using different weapons on each level.

Usability
The title screen is incomplete and consists of a background and a start button. There are no game instructions. Left and right arrow keys are used to control player character movement and the space bar is used for firing missiles. Three levels are linked thematically but level 3 is not functional. The game features one animated sprite. Another animated sprite is loaded in the game resources but is not implemented in the game.

Functionality
There is limited functionality. The start button launches the game. The player can move the main character left and right using the arrow keys, although the character does not stop moving on key release and travels off the screen; he can be returned to view if the arrow keys are pressed. The player can launch an animated stone using the space bar. If the stone hits a monster, the monster disappears but no score is implemented. The monsters do not move and do not reappear once destroyed. On level 2 the player can move the main character left and right as before and launch stones, but no monsters are implemented.

Scoring
The score is set at the start of the game but is not further implemented, so points are neither gained nor lost and no score is displayed on screen.

Gameplay
The player can control the movement of the main character and can throw stones at monsters, which disappear when they are hit. There is no win/lose state or level progression.

Sound
No sound implemented

Game design
The game backgrounds are visually coherent. Background images, weapons and enemy objects differ on each level.

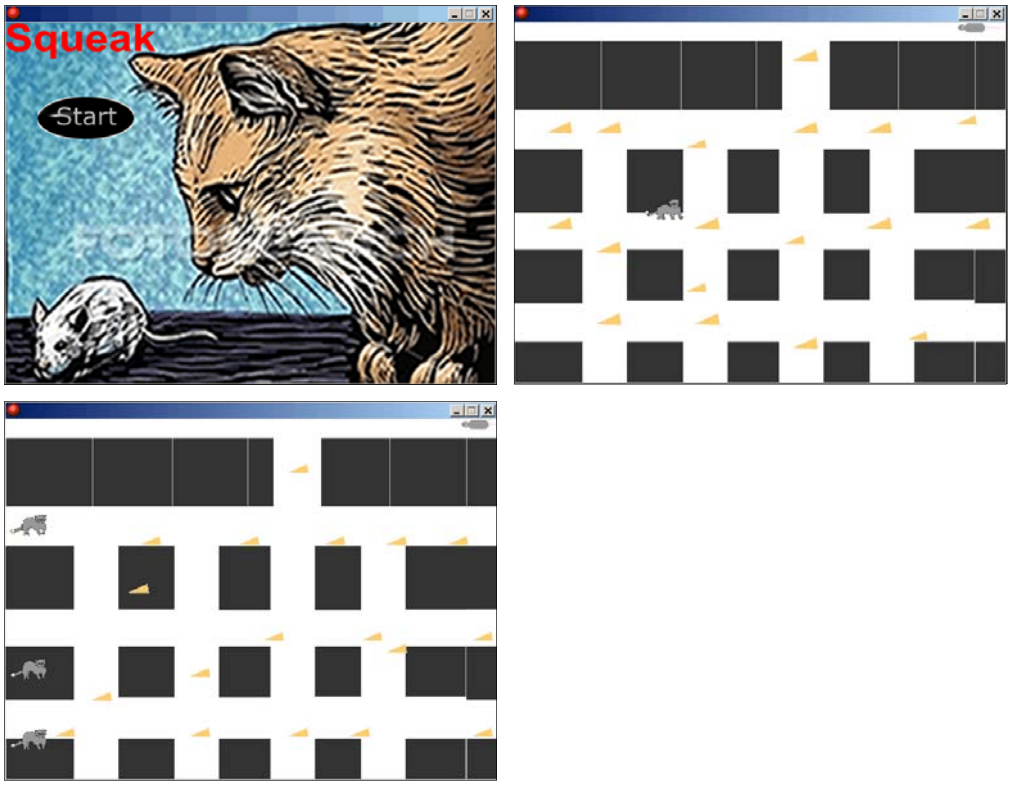
Programming
The game evidences that the boys understood the concept of event-driven programming - they correctly use some events as input data. Sixteen events and 18 actions are used to create the gameplay. There is no use of conditional statements. The *step* event is used to create monsters at intervals, repeatedly. They showed some awareness of the need to use a variable to store score data, but did not implement the scoring mechanism fully. They also set object speed variables. Although they included actions which required the setting of coordinates, values were not set for these. They applied negative number in setting the vertical speed of the stone object to specify an upwards movement.

Graphics
This pair used a mix of graphics they created themselves and those they sourced online (player character, ghosts). Their own graphics (brick, bottle, start button) are oversized and not well executed. Background images used in the game levels are effective.

The matrix below maps the SOLO score achieved for each component:

Extended	10								
	9								
Relational	8								
	7								
Multistructural	6								
	5								
Unistructural	4								
	3								
Pre-structural	2								
	1								
		Usability	**Functionality**	**Scoring**	**Gameplay**	**Sound**	**Overall design**	**Programming**	**Graphics**

LWGW
KS2 SAT average 4.99/5.26; CAT average 95/102; Jesson band above average/high



LWGW title screen, level 1, level 2

This game ranks 9th with a score of 25/80.

Code organisation and documentation
9/12 planning documents were completed. 12/16 sprites, 10/14 objects and 3/4 backgrounds are named correctly. Rooms are incorrectly named and 2 room names are duplicated. 1/1 sound is unnamed. The 'start' button sprite is loaded twice, and unnamed once. The title screen is loaded as a sprite, twice, as well as a background. One background is loaded as an object. There are some redundant items.

Problems
The main problem in this game is that there is no mechanism to progress from one level to the next. It is difficult for the player to control the movement of the player character (a mouse) - there is no event to stop its movement so it eventually travels off screen. The value for the mouse's speed is set to relative, which means that it moves increasingly quickly and becomes impossible to control. When the *down* arrow key is pressed the mouse drops out of view. The enemy object (a cat) is set to move in random directions and eventually disappears from view. Although this game is intended to be a maze game, the maze walls are drawn as part of the background graphic and not created as solid objects so they do not function as a boundary. Because these girls did not name sprites and objects correctly it was difficult for them to reference these correctly in events and actions, which caused problems in the game (e.g. the wrong sprite was loaded or the wrong object is defined in *collision* events).

Game storyline
In *Squeak*, a mouse runs through a kitchen, eating cheese to gain points and avoiding cats. If caught by a cat, the mouse loses a life.

Usability
The title screen offers a 'start' button but there are no game instructions. Arrow keys are used to control the directional movement of the player character. There is no score mechanic. There is one animated sprite, but the player character does not change appearance on direction change. No mechanism for progressing through the 3 levels is implemented.

Functionality
There is limited functionality. The 'start' button launches the game. The player has some control of the main character using the arrow keys but there is no mechanism to stop movement. There are limited interactions between the mouse and the cheese objects, only some of which disappear when 'eaten'. The enemy objects (cats) move randomly but there are no interactions between these cats and other objects.

Scoring
No score implemented.

Gameplay
The player has limited control over the player character and no control over any other objects apart from the 'start' button on the title screen. The player character can eat cheese (some of which disappears). There is no score, no penalties and no win/lose state. No mechanism for progressing through the levels is implemented so there is no challenge in the game.

Sound
One sound is loaded in the game resources but not implemented.

Game design
Although designed as a maze game, the game does not function as such. The game is coherent narratively and visually, but there is little variation in level design or object inventory across the 3 levels.

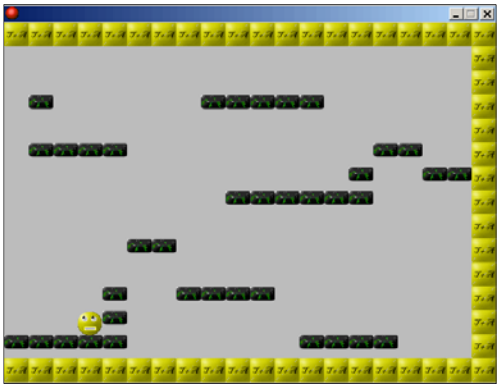
Programming
These girls use some events correctly (*create, collision, key press, mouse*) but others are misunderstood. Fifteen events and 23 actions are used to create the gameplay. No conditional statements or loops are used. Variables are not used to store data in the game. There is no use of logical operators, Boolean logic, relational operators, coordinates or negative number. Randomness is used to define the cat's movement. Relative value is used in error to set the player character's speed.

Graphics
These graphics in this game (an animated cat, mouse, cheese, background, start button) are poorly executed. The image on the title screen was sourced online.

The matrix below maps the SOLO score achieved for each component:

Extended	10								
	9								
Relational	8								
	7								
Multistructural	6								
	5								
Unistructural	4								
	3								
Pre-structural	2								
	1								
		Usability	**Functionality**	**Scoring**	**Gameplay**	**Sound**	**Overall design**	**Programming**	**Graphics**

AWTB
KS2 SAT average 4.22/5.34; CAT score average 97/113; Jesson band below average/high



AWTB level 1

This game ranks 10th with a score of 24/80.

Code organisation and documentation
7/12 planning documents were partially completed. 1/12 sprites is named correctly; one is unnamed; 0/9 objects are named correctly with obj_ prefix; one object is unnamed.

Problems
The main problem with this game is that there are limited game interactions. Designed as a platform game, the player character's jumping mechanic is not well implemented; the object disappears from the bottom of the screen when it falls from the lower platforms and cannot be returned to view. Although the background scrolls, the scrolling movement is not apparent because there are no background graphics. No score or lives mechanics are implemented.

Game storyline
Don't Stop Running is a vertically scrolling platform game. In their original ideas the story involved an escaped robber being chased by a policeman through an Australian cityscape but this narrative is not discernible in the game implemented.

Usability
No title screen is implemented. 'Start', 'instructions' and 'exit' buttons are created but not implemented. There are no game instructions. Common control options are used to control the movement of the player character (arrow keys, space bar to jump) on one level. There is no user feedback.

Functionality
There is limited functionality - the player can make a character jump and move left and right to land on platforms. But the jumping mechanic is not consistent - when the space bar is pressed the character jumps erratically. No score or lives mechanics are implemented. There are no penalty/reward objects to interact with.

Scoring
No score implemented.

Gameplay
The player can make the main character jump and land on platforms - the goal is to get the character to jump to the highest platform without falling out of the bottom of the room and this provides some challenge. There are no other interactions possible.

Sound
No sound implemented.

Game design
The game is not sufficiently developed to have structural coherence. There is one level but the game has no visual appeal. Some objects are not implemented.

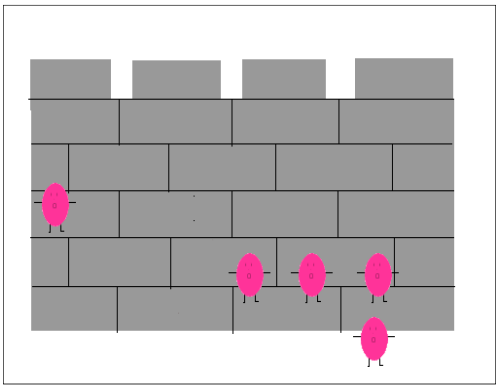
Programming
This pair showed some understanding of event-driven programming and used 6 events and 40 actions to create the game play. They used 9 conditional statements to *check position*, to *check collision* and to *test variable* (x,y). One *step* event is used to check whether the player character is 'in the air' and to limit its vertical speed. Variables are used to *set vertical speed, set speed, set gravity, set variable* (x,y)*, test variable* (vspeed). Boolean logic is used to define objects as solid. A relational operator (>) is used to test speed. Coordinates are used to check position. Angles are used to define gravity and direction. Negative number and relative value are used to define position.

Graphics
The graphics in this game are poorly executed. Title, title screen background, start, and exit buttons, and crate objects were created but not implemented.

The matrix below maps the SOLO score achieved for each component:

Extended	10								
	9								
Relational	8								
	7								
Multistructural	6								
	5								
Unistructural	4								
	3								
Pre-structural	2								
	1								
		Usability	**Functionality**	**Scoring**	**Game play**	**Sound**	**Overall design**	**Programming**	**Graphics**

SARC
KS2 SAT average 4.7/5.37; CAT score average 100/114; Jesson average/high



SARC level 1

This game ranks 11th with a score of 14/80.

Code organisation and documentation
6/12 planning documents were completed. 2/4 sprites and the background are correctly named. The room is unnamed.

Problems
The main problem in this game is that there is no user interaction. The player character is not implemented, so no interactions in the game are possible. Because these girls did not establish a clear storyline for their game and did not make use of their plans it was difficult for them to translate their ideas into game action.

Game storyline
Starman fights enemies as he moves through a castle to collect gold coins and gain points.

Usability
There is no title screen, no user options and no game instructions. Common control options (arrow keys) are used to control the movement of the player character but these are not implemented correctly. The game consists of one level.

Functionality
No interactions are implemented. On game start, 9 instances of the enemy object move randomly and disappear from the screen after 10 seconds of game play. The player character is not placed in the game room so is not visible on the screen.

Scoring
No score implemented.

Gameplay
There is no user interaction and no game play.

Sound
No sound implemented.

Game design
There is no structure to the game. The characters have no discernible association with the setting of the castle.

Programming
These girls understood the concept of event-driven programming and used *keyboard* events with the intention of controlling directional movement of the player character, but no movement actions were added. The *create* event is used to set the random movement of enemy objects. Five events and 5 actions are used to create the gameplay. No conditional statements are used. There is limited use of variables - speed is set correctly for the enemy object, but used without understanding in the *change sprite* action.

Graphics
The graphics in this game are not well executed.

The matrix below maps the SOLO score achieved for each component:

Extended	10								
	9								
Relational	8								
	7								
Multistructural	6								
	5								
Unistructural	4								
	3								
Pre-structural	2								
	1								
		Usability	**Functionality**	**Scoring**	**Game play**	**Sound**	**Overall design**	**Programming**	**Graphics**

Appendix 2a: Pupil information booklet

YEAR 9 ICT

Computer Game Authoring

Pupil Information

Year 9 ICT Computer Game Authoring Unit

Pupil Information Sheet

The computer game industry is one of the fastest growing sectors of the economy and some games have become blockbuster entertainment. Most of you will have played computer games - but very few of you have ever created a computer game…until now!

This year a Game Authoring unit will be included in Year 9 ICT lessons, to give you an opportunity to learn about and to design a computer game for the first time, so that you don't just play games, but get a chance to make them! The unit is part of the National Curriculum for ICT at Key Stage 3.

It is important that you have some experience of computer game authoring at Key Stage 3 because increasingly, examination boards are developing specifications aimed at Key Stage 4 and beyond, which include game authoring units - so next year, you may be able to study Computer Game Authoring as one of your options, if you choose to do ICT.

Year 9 Research Project

I have recently been trialling game authoring software with Years 7 and 9, to see which is the best program to use. Now I would like to research what Year 9 pupils themselves think of computer game authoring.

I would like us to begin this project in the spring or summer term. You will research, plan and design a computer game with a partner.

What you will learn

You will learn about game narrative development, logical thinking, graphics, programming, sequencing instructions, music making/editing and how to test your game. You will use a range of software during the project (Game Maker, Macromedia Fireworks, Audacity, Microsoft Word, Internet Explorer, Dance E-Jay).

What I will be researching

I would like to conduct my research with your ICT class as part of a study that I am completing at the University of Southampton. I am interested in finding out about the benefits of computer game authoring, both as a creative and an intellectual activity.

As part of my research, I would like to record you talking as you work in pairs to plan, design, build, test and evaluate your computer game, during your normal ICT lessons. I will also be asking you to complete a journal about your learning experiences for homework. I will interview some pupils in a group and/or in pairs.

The work you complete during the project will be analysed and I may publish my findings in various ways (i.e. at conferences, in journal articles, and on the internet).

What do you have to do?

You have to complete all the tasks in the Game Authoring Unit, as part of your ICT lessons. I will mark your work and give you a National Curriculum level, as normal. The only difference is that I will be using your work to help me find out about how you learn to create a computer game. You do not have to put your name on anything, so all your work will be anonymous. All of your work will be stored securely on the school network and analysed on my personal computer at home. If you are happy for me to use your work, anonymously, as part of this research project, talk to your parents/carers about it and, with their agreement, sign the form on the next page. You can change your mind even after you have signed the form.

If you do not want your work to be included in my research, you will not be at a disadvantage – you will follow the same scheme of work and be given the same opportunities to create a computer game as those who do take part in the study.

Thank you very much if you are able to support my research by completing the form overleaf.

Ms Johnson, Subject Leader, ICT.

Year 9 Game Authoring Project

Please circle 'Yes' if you agree that your work can be used in the research. Circle 'No' if you do not want your work used. If you tick 'Yes' to any of the statements this does not guarantee that your work will be used or that you will be interviewed.

Please return the form in the envelope provided.

Your name: _____ **Tutor:** _____

I understand that if my work is used, it will be anonymous (no-one will know it is my work).	**Yes**	**No**
I agree that my work can be used as part of the research project.	**Yes**	**No**
I agree that my digital voice recordings can be used as part of the project.	**Yes**	**No**
I agree that my journal can be used as part of the project.	**Yes**	**No**
I agree that my computer game can be used as part of the project.	**Yes**	**No**
I agree to being interviewed as part of a group of six pupils.	**Yes**	**No**
I agree to being interviewed in a pair.	**Yes**	**No**
I agree to my work being shared with other people i.e. in reports, conference papers, journals and on the internet.	**Yes**	**No**
My parent(s)/carer(s) agree that my work can be used in the project.	**Yes**	**No**

Signed: _____

(Pupil)

Appendix 2b: Parent information and consent form

Dear Parent

Re: Year 9 ICT Research Project in Computer Game Authoring.

The computer game industry is one of the fastest growing sectors of the economy and some games have become blockbuster entertainment. Most pupils have played computer games – but very few have ever created a computer game…until now!

This year, a Game Authoring unit will be introduced into the Year 9 ICT curriculum, to give pupils an opportunity to design a computer game, so that they are not just consumers, but also producers of this medium. This is part of the National Curriculum for ICT at Key Stage 3 and follows the learning objectives of the revised programme of study. A range of software will be used during the project (Game Maker, Macromedia Fireworks, Microsoft Word, Microsoft Internet Explorer, Audacity, Dance E-Jay).

Computer game authoring is an important addition to the Key Stage 3 ICT curriculum because it gives pupils a creative, real world scenario in which to develop their skills in computer programming, graphics, logical thinking, narrative development and so on. It is important to deliver this unit of work at Key Stage 3 because increasingly, examination boards are developing specifications aimed at Key Stage 4 and beyond, which include computer game authoring units.

I have recently been trialling game authoring software with Years 7 and 9, to see which offers the most successful learning outcomes for all pupils. Now I would like to conduct some formal research with Year 9 pupils, to find out what they think of computer game authoring.

I would like to conduct this research with your son/daughter's ICT class as part of a case study that I am undertaking, to be supervised by the University of Southampton's School of Education. In this study I will investigate the value to pupils of computer game authoring, both as a creative and an intellectual activity.

As part of this research, I would like to record pupils' talk as they work in pairs to plan, design, build, test and evaluate their computer game, during their ICT lessons. I will also ask pupils to complete a journal of their learning experiences for homework. I will interview some pupils as part of a group, and in pairs.

Material generated by pupils during the project will be analysed and research findings will be published in various formats and may be shared with third parties (i.e. at conferences, in journal articles and on the internet).

All pupils participating in the study will be able to withdraw their data at any time and their anonymity will be preserved throughout. All data will be stored securely on the school network and analysed on my personal computer at home.

If you are willing to give your consent for me to use your child's work, anonymously, as part of this research project, please complete and return the form below.

If you do not wish your child's work to be included in the research, they will not be at a disadvantage – they will follow the same scheme of work and be given the same opportunities to create a computer game as those who do take part in the study.

Thank you very much if you and your child are able to support this research by giving your consent on the form below.

Please do not hesitate to contact me if you would like to know more.

Yours sincerely

Claire Johnson
Subject Leader, ICT

Year 9 Computer Game Authoring Research Project

Please read the statements below and tick the boxes as appropriate. NB This does not guarantee that your child's work will be used or that they will be interviewed as part of the study.

Please return in the envelope provided.

Name of pupil: _____ Tutor: _____

	Yes	**No**
I understand that my child's anonymity will be preserved in all items below.	☐	☐
I give my consent to my child's work being used as part of the research.	☐	☐
I agree to my child's digital voice recordings being used as part of the study.	☐	☐
I agree to my child's journals being used as part of the study.	☐	☐
I agree to my child's computer game being used as part of the study.	☐	☐
I agree to my child being interviewed as part of a group of six pupils.	☐	☐
I agree to my child being interviewed as a pair.	☐	☐
I agree to my child's work being shared with third parties (i.e. in reports, conference papers, journals and on the internet).	☐	☐

Signed: _____

(Parent/Guardian)

Appendix 3a: Interview schedule – group interviews

Introduce the interview, its purposes and how the data will be used. Assure confidentiality. Explain that pupils don't have to answer questions directly to you - they can discuss their responses with each other. Display Game Maker on the whiteboard. Explain that you are interested in what they think about the game making process - there are no right or wrong answers etc.

1. Tell me about your experience of game making so far?

2. What did you think about using the video tutorials to learn the software skills?

3. How else would you have liked to learn about how to use *Game Maker* (e.g. textbooks, Moodle course, me showing you on the board, working through paper based tutorials on your own, other)?

4. You have been working on your games in pairs. How has working with a partner helped you to learn about making a computer game? Were there any problems with working in a pair?

5. What about the *Game Maker* software?

6. When you learn something new, sometimes it's a bit hard to begin with. Tell me about any difficulties you had. This may be to do with the software or it may be to do with the game authoring process.

7. This is the first time you have ever made a computer game as part of your school work. How did you decide or know what you needed to do?

8. You have been making your game for several lessons now. What do you think you have learned so far?

9. Creating a game which involves the player interacting with characters and objects on the screen is probably a new thing for you. What do you think it teaches you that is different from what you learn in other ICT projects?

10. The real work in this project is in making things happen on screen and making sure that everything works as it should. Can you think of one time when you were having problems with something? What was it and how did you solve the problem?

11. What do you think is most successful about your game, even if it isn't finished yet? Have you managed to do anything you are particularly pleased with?

12. What things are hard to understand when making a game? What sorts of things are you getting stuck on?

13. I have been asking you to record yourselves as you make your game. How have you found this?

14. When you make a game you are creating an interactive system, where you have to make things happen to create good game play. You also have to make it look attractive and it has to be intuitive for the player to play. This is a bit

different to making a leaflet or designing a logo or creating a spreadsheet or a presentation. What do you like about this kind of work?

15. Anything you don't like about it?

16. How do you think making a game makes you think differently, compared with the other things you have done in ICT?

17. How does making a game challenge you or stretch you or make you think?

18. Some people might say that making games is not a serious topic and it doesn't teach you anything. What do you think about this?

19. One of the things you have learned which is new to you is the vocabulary of game authoring, - words like 'sprite', 'event', 'action', 'object', 'relative', etc. Can you think of anything else new that you have learned in this project?

20. This project took up 16 lessons. What do you think about the time scale? Too, long/short?

21. If you were to do a game making project again, can you think of any improvements or changes you would make to the way things were done?

22. If you had to tell a friend about this project, what would you say?

Appendix 3b: Interview schedule – paired interviews

Introduce the interview, its purposes and how the data will be used. Assure confidentiality. Explain that you are interested in what they think about the game making process and hope to learn more about their thinking/learning as they created their game. Assure pupils that there are no right or wrong answers. Load games.

1. Tell me about your game.

2. How has working as a pair helped you to learn about making a computer game?

3. How did you decide as a pair what needed to be done each lesson?

4. How did you work out what objects, events and actions you would include in your game?

5. Tell me about any difficulties you had:

    a. With the game making process
    b. With *Game Maker*

6. What do you think you have learned in this project?

7. What do you think it teaches you that is different from what you have learned in other ICT projects?

8. What do you think game making helps you to learn?

9. Can you think of a time when you were having problems with something? What was it and how did you solve the problem?

10. Making a game is different from other ICT tasks. What do you like about this kind of work?

11. Anything you don't like about it?

12. How does game making make you think differently, compared with the other things you are used to doing in ICT?

13. How does game making challenge you or stretch you or make you think?

14. Does it teach you any new skills that you haven't used so far?

15. What things are hard to understand with making a game?

16. If you had to evaluate your game and say what is successful and what needs improvement, what would you say?

Appendix 4: Prompt sheet for digital voice recordings

Recording your work in progress.

'What do I say?' you may ask when you hit the record button! You haven't done this before so it may seem a little strange at first. Sometimes it can be hard to think of what to say when you know you are being recorded! Try not to worry about this - as long as you are talking about the work you are doing that will be just fine!

I want to find out how you go about making a computer game - so whatever you have to say is interesting to me - as long as it is to do with your work! There are no right or wrong answers - and as you get more used to recording yourselves, it will be less of a problem for you.

Here are some ideas to prompt you if you get stuck for words!

Talk about the objects in your game and how you decide what events and actions to give them.

Talk about the settings you select for the actions you decide to use.

Talk about any problems or difficulties you are having.

Talk about what is going well and what you are pleased with.

Appendix 5: Data coding

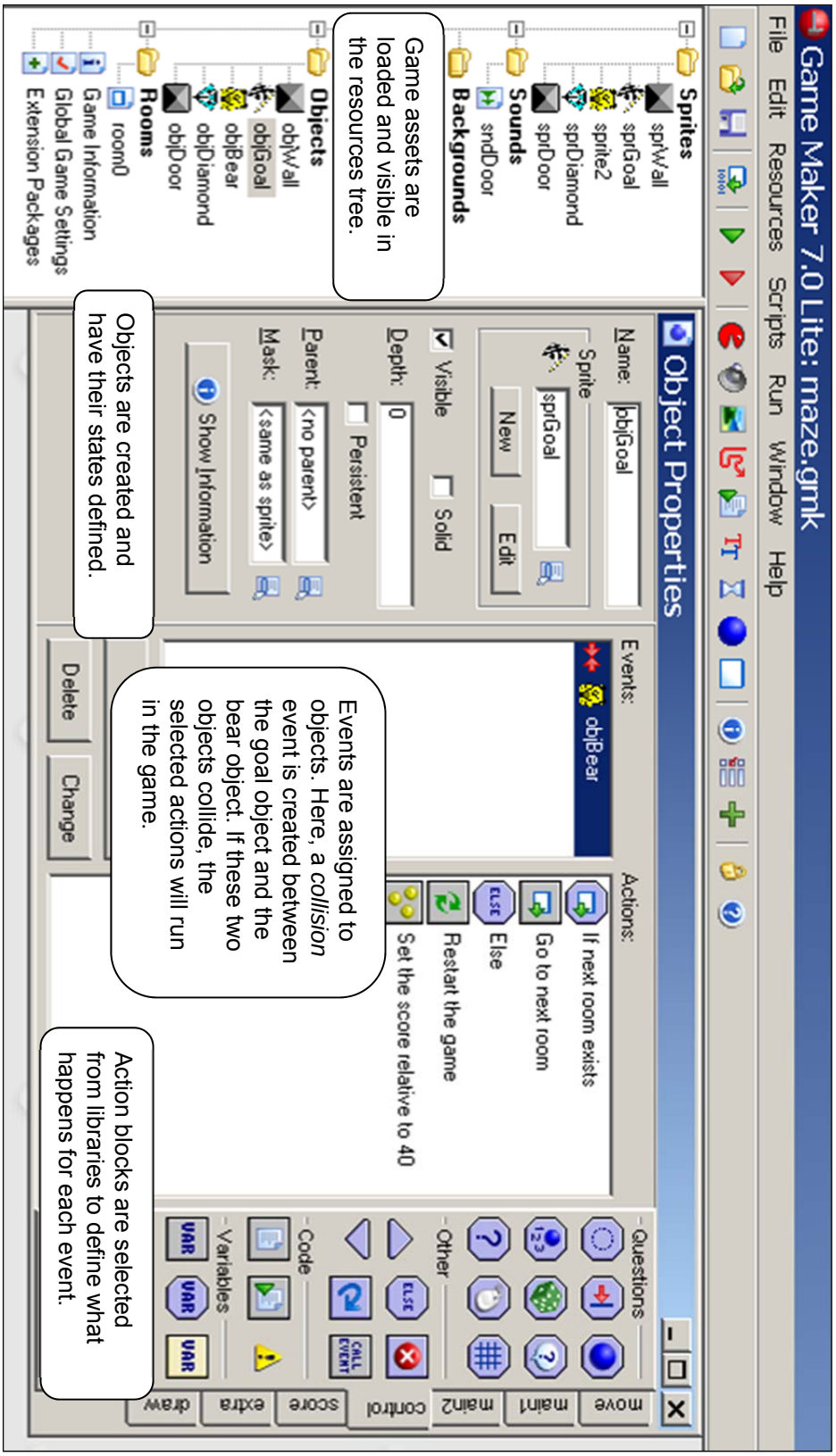
Free nodes (initial codes) used to code the transcript of the digital voice recordings, group and pair interviews and pupil documents.

Category	Code	Description
Gender	*M/F*	Assign gender as an attribute in NVivo
Ability	*Low* *Below average* *Average* *Above average* *High*	Assign Jesson level as an attribute in NVivo
Research questions	*Process*	RQ 1a) References to pupils' perceptions about the process of their learning
	Outcome	RQ 1b) References to pupils' perceptions about the outcomes of their learning
	Programming	RQ 2) Use of computer programming constructs and terminology e.g. 'variable', 'IF statement', sequence
	Design difficulties	RQ 3a) References to difficulties pupils have with game design
	Programming difficulties	RQ 3b) References to difficulties pupils have with game programming
	Value	RQ 4) References to affective values of making computer games
Conceptual Framework 8 big ideas of constructionism concepts	*Doing*	Learning by doing
	Making	Technology as building material
	Learning	Learning to learn
	Freedom	Freedom to get things wrong
	Time	Taking time
	Teacher	Teacher as co-learner
	Hard fun	'Hard fun'
	Computers	Using computers to learn in a digital world
Game Maker language	*GM*	Use of words specific to game design concepts in *Game Maker* e.g. 'sprite', 'object', 'relative', 'solid', 'room height'
Game design	*Design*	References to design concepts e.g. challenge, sound
	Representations	Cultural references and representations
	Narrative	References to narrative, characters, storyline
Use of software	*Software*	References to the use of software
Mode of learning	*Mode*	References to mode of learning e.g. trial and error, tutorials, learning from others, working in pairs etc.
Mathematical concepts	*Maths*	References to mathematics concepts e.g. use of coordinates, randomness, gravity
Graphics concepts	*Graphics*	References to graphics concepts e.g. transparency, size, animation etc.
Experience of the activity	*Experience*	References to their experience of game authoring e.g. freedom, creativity, fun
Attitudes	*Attitudes*	References to attitudes to learning, engagement, persistence, independence, commitment, etc.
Evaluation	*Evaluation*	Comments which evaluate their games

Free nodes used to code the programming difficulties evidenced in the analysis of authored games.

Category	Code	Description
Programming difficulties	*Actions*	Errors relating to the use of actions
	Angles	Errors relating to the use of angles
	Background	Errors or problems relating to the game background e.g. scrolling
	Conditionals	Errors in the use of conditional statements (test/check actions).
	Conflicting actions	Use of conflicting actions
	Conflicting events	Use of conflicting events
	Coordinates	Errors in the use of coordinates
	Duplicate	Duplicating events/actions
	Logic	Errors which occur due to 'illogical' thinking
	Events	Errors relating to the use of events
	Incomplete	Errors which arise because a construct is incomplete
	Levels	Errors relating to levels in the game
	Lives	Errors relating to the lives mechanic
	Objects	Errors relating to objects
	Misc	Miscellaneous errors/problems
	Movement	Errors or problems to do with object motion
	Negative number	Errors relating to the use of negative number
	Random	Errors relating to the use of 'randomness'
	Redundant	Use of redundant (i.e. unnecessary) code
	Relative	Errors in applying the 'relative' parameter
	Score	Errors or problems with the score mechanic
	Screen boundary	Errors relating to objects travelling off screen
	Self/other	Errors or problems with selecting self/other e.g. in collision events
	Sequence	Errors or problems relating to the sequencing of events/actions
	Solid	Errors in applying the 'solid' parameter
	Sound	Errors relating to the playback of sound
	Speed	Errors relating to object speed
	Values	Errors relating to the assignment of values in an action or expression (e.g. value for speed, coordinates)
	Variables	Errors relating to the use of variables

Appendix 6: The *Game Maker* interface



Appendix 7: Outline scheme of work

Learning objectives	Lesson activities
Lesson 1 Identify the main components of a computer game. Recognise the potential of the *Game Maker* software.	**Starter** Introduction to the project. **Main** Play and evaluate sample games (maze, platform, shooter, action etc.). Identify and evaluate key components of a game/elements of game play (worksheet). **Plenary** Discuss with your partner what ideas you have for your own game.
Lesson 2 Develop skills in the *Game Maker* software. Review the range of computer game genres.	**Starter** Teacher demo: intro to *Pixel 8* video tutorials, how to load the game resources etc. **Main** Complete the 'Catch the Piggie' video tutorial. **Plenary** Complete the 'Genres' worksheet. **Homework** Some people say that the best way of learning is by doing. Write about your experience of using the video tutorials. What are your impressions so far of the *Game Maker* software? Any other comments?
Lesson 3 Further develop *Game Maker* skills. Understand the importance of developing a game storyline.	**Starter** Complete 'The *Game Maker* interface' worksheet. **Main** Complete the 'Maze Game' video tutorial. **Plenary** Read the 'Game Storyline' example sheet. Select a preferred genre (maze, platform, shooter, action, breakout) and discuss with your partner your initial ideas for a game, using the project brief sheet. **Homework** Develop your initial ideas for a game storyline and characters by creating a storyboard of the main action/rooms in your game, including title and end game screens.

Lesson 4 Identify the main objects, events and actions in your game.	**Starter** In your pairs agree the final storyline for your game. Complete the 'Initial Design' sheet. **Main** Complete the 'Space Shooter' video tutorial. **Plenary** View the 'How is a computer game made?' video.
Lesson 5 Review *Game Maker* terminology. Understand the importance of a game design document. Develop success criteria for a game.	**Starter** Complete the '*Game Maker* Terminology' worksheet. **Main** View the 'Game Design' presentation. Read the design document for 'Catch the Clown'. Complete a design document for your own game. **Plenary** What makes a good game? Develop 10 success criteria. **Homework** Now you have devised a storyline for your game and have written a design document for it. Write about how you decided on your game ideas. Did you have any problems with this? Any other comments?
Lesson 6 Review the ready-made sprite assets in *Game Maker.* Learn to use the *Game Maker* image editor to edit/animate sprites.	**Starter** Teacher demo: the *Game Maker* sprites library. **Main** Create or locate graphic objects (sprites, backgrounds, other game objects) for your game. Use *Fireworks* or the image editor in *Game Maker*. **Plenary** Teacher demo: animated sprites/changing a sprite's image.
Lesson 7 Review available events and actions in *Game Maker.* Practise 'reading' and modifying *Game Maker* 'code'.	**Starter** Learn to read *Game Maker* 'code' - complete the '*Game Maker* Programming' sheet. **Main** Modify the code in a *Game Maker* sample game. Show another pair the modifications you have made. **Plenary** Develop a 'systems development life cycle' for a game - see p100-1 *ICT 4 Life*.

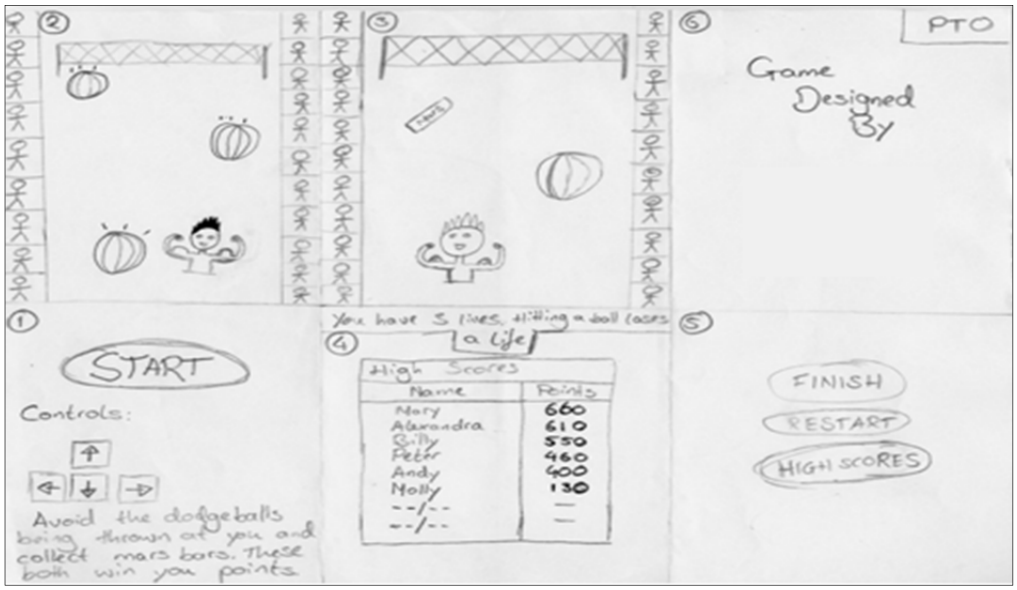
Lesson 8 Understand events. Understand actions. Identify the rules for your game.	**Starter** View the 'Events and Actions' presentation. **Main** Teacher demo: events and actions in *Game Maker* sample games. Read the 'Rules of the Game' example sheet. Complete the 'Rules of the Game' worksheet for your game. **Plenary** Teacher demo: the 'Show object information' item in the *Game Maker* 'Edit' menu. **Homework** Deciding on the rules for your game is very important - you have to decide on how the player interacts with objects in the game and control what else happens in the game. It's what makes a game fun to play! Tell me about how you did this.
Lesson 9 Understand the use of rooms/levels.	**Starter** Teacher demo: multiple rooms/levels in a sample game. **Main** Create one or more rooms for your game. Begin to add objects to your game and specify their events and actions. **Plenary** Teacher demo: moving from one room/level to the next. **Homework** You have learned how to create rooms/levels for your game. Write about how you did this. Describe any problems or difficulties you had with this. Any other comments?
Lesson 10 Understand the use of the *test variable* action in *Game Maker.*	**Starter** Write the instructions for a computer version of 'Noughts and Crosses' using the 'If…then' construction. (*ICT Interact Year 8* p19). **Main** Teacher demo: *test variable* action - If…then. Develop your game. **Plenary** Explain how you have used or how you could use a *test variable* action in your game.

Lesson 11 Understand *step* events. Develop game mechanics.	**Starter** Review the game mechanics of *Game Maker* sample games. **Main** Teacher demo*: step* events. Add a *step* event in your game. Develop your game. Focus on the mechanics of your game (score, lives/health, collisions, scrolling). **Plenary** Explain how you have made use of one game mechanic in your game to improve game play. **Homework** You have been learning how to control what happens in your game by adding events and actions to objects. Write about the events and actions you have used in your game. Describe any problems or difficulties you had with this. Any other comments?
Lesson 12 Consider the use of sound in a game.	**Starter** Teacher demo: the use of sound in *Game Maker* sample games. **Main** Consider the use of sound in your game. Locate/create sound effects and background sounds for your game. Develop your game. **Plenary** Invite another pair to evaluate the sound you have used in your game. **Homework** Sound is an important element of any game. Tell me about what sounds you chose to use. What effect were you trying to create by using these sounds? How did you create/locate your sounds? Did you have any problems with this? Any other comments?
Lesson 13 Understand the role of textual information in a game.	**Starter** Teacher demo: title/end game/instructions screens in *Game Maker* sample games. **Main** Develop the title/end game screens for your game. Create these in *Fireworks*. Develop the 'game information' and 'instructions' for your game. Add a *message* action to a suitable event. **Plenary** Invite another pair to test and evaluate your title screen and game instructions.

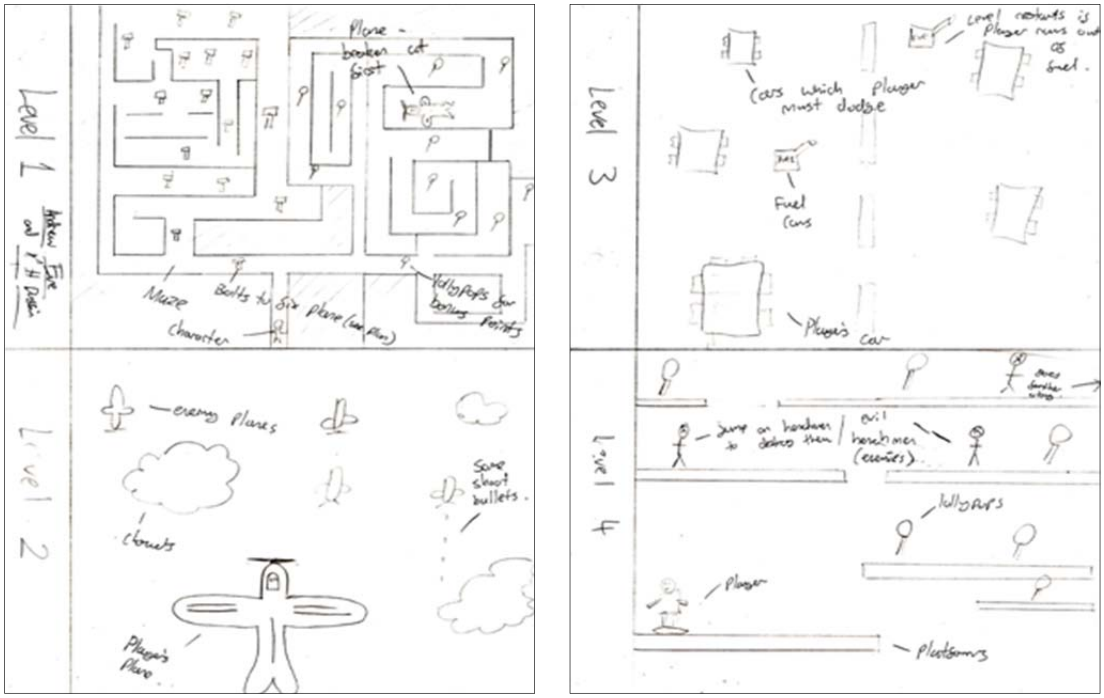
Lesson 14 Understand the importance of testing your game.	**Starter** View the 'Testing your game' presentation. **Main** Develop a test plan for your game and test it. Develop your game, making amendments as necessary. **Plenary** Test another pair's game and suggest one improvement.
Lesson 15 Create an .exe version of your game.	**Starter** Teacher demo: how to create an .exe version of your game. **Main** Implement final changes to your game. Create an executable version of it for others to play on any PC. Upload your game to the *Game Maker* website? **Plenary** Demonstrate your game to another pair and invite them to evaluate it. **Homework** Begin to evaluate your game. Assess what is effective and what needs improvement.
Lesson 16 Evaluate your game using established success criteria.	**Starter** Review your success criteria for a good game (Lesson 5). **Main** Complete your game evaluation. Use the project evaluation worksheet or record a spoken evaluation. **Plenary -** **Homework** If you had to tell someone about what you have learned during the project, what would you say?

Appendix 8: Pupils' storyboards

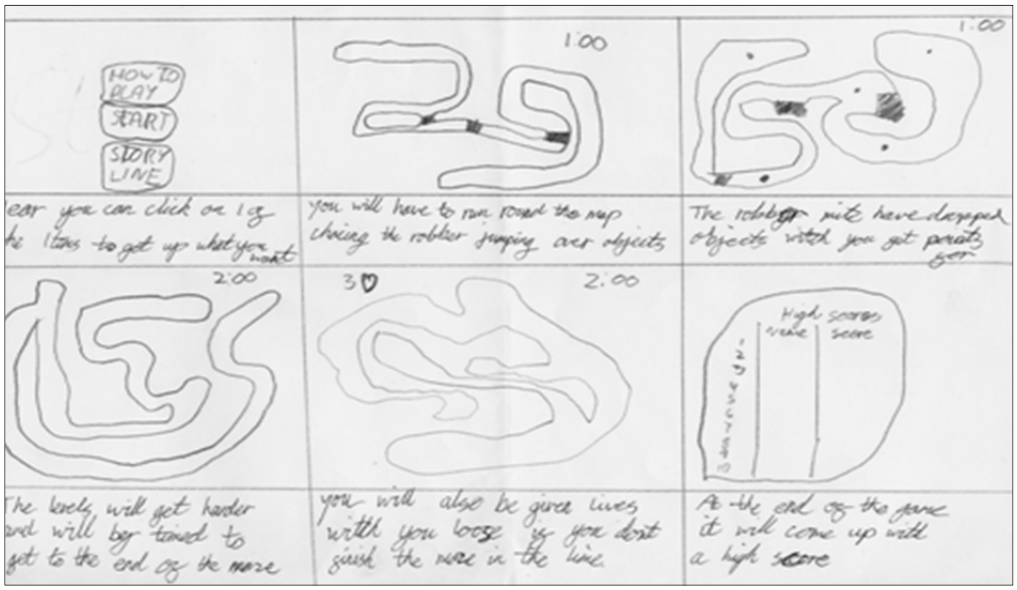
This appendix supplements section 6.3.1 above, which describes the features included in and missing from pupils' storyboards, completed for homework as part of the planning for their computer games.



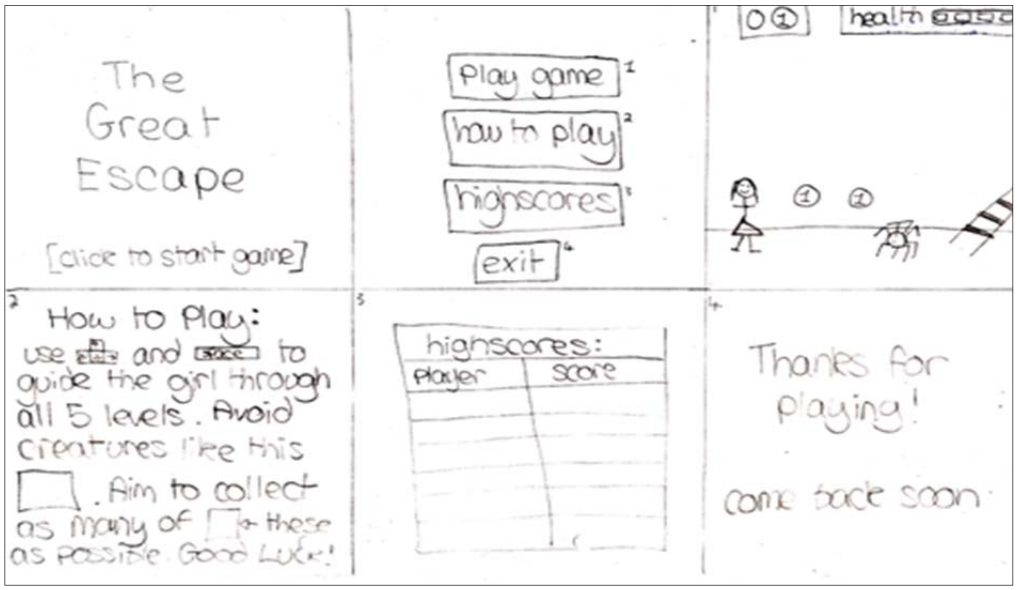
ACJC's storyboard



AEMD's storyboard



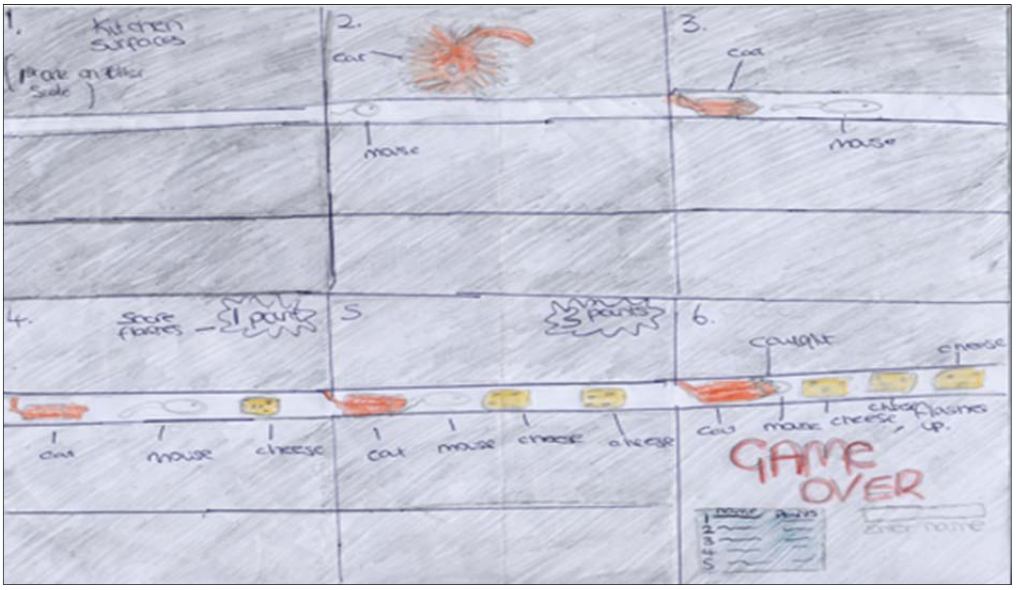
AWTB's storyboard



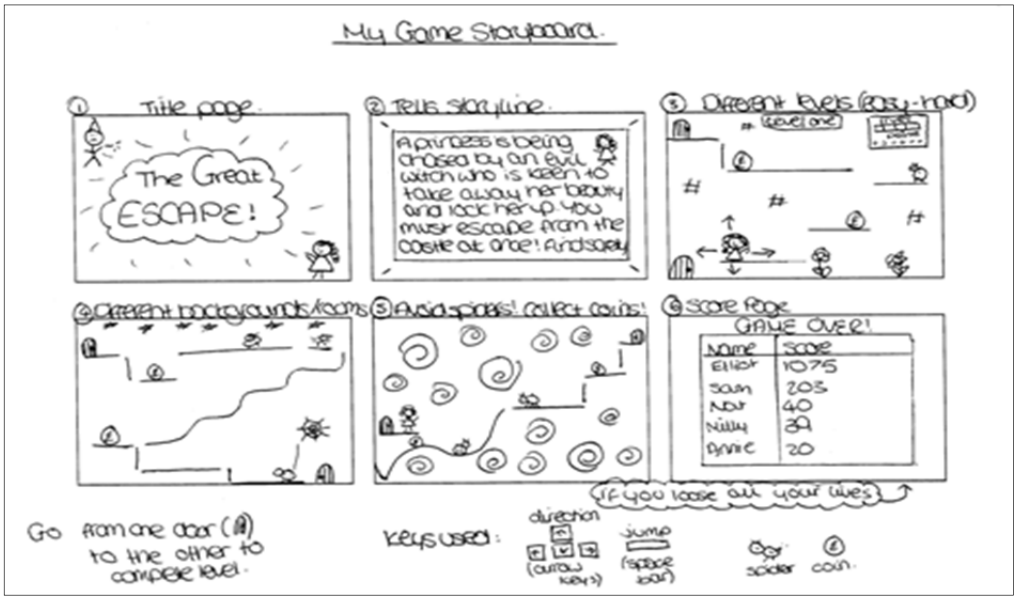
CB's storyboard

KW's storyboard

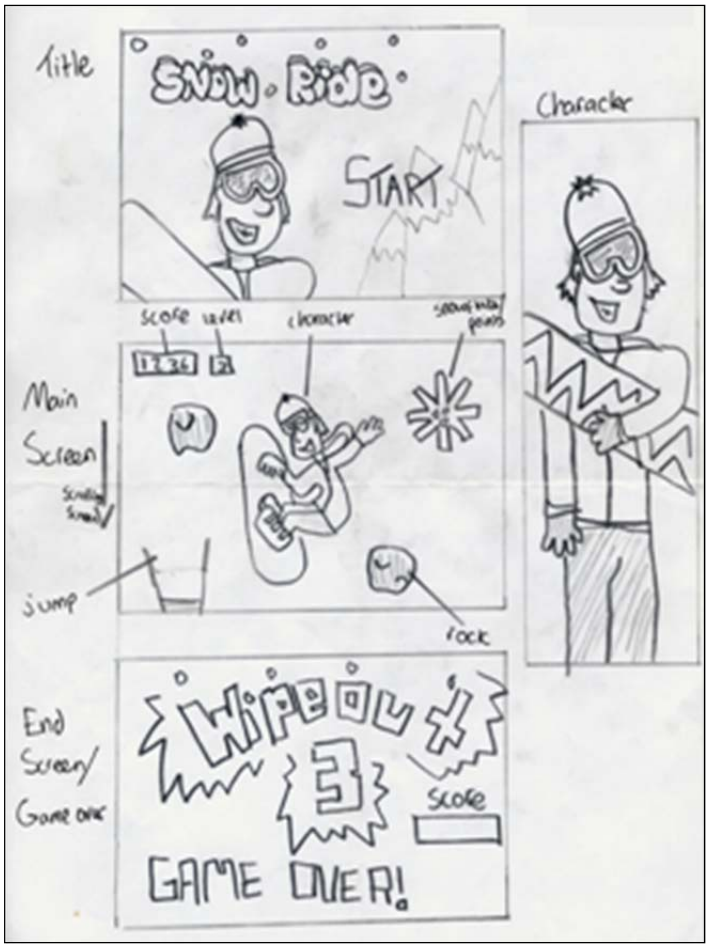
LWGW's title screen



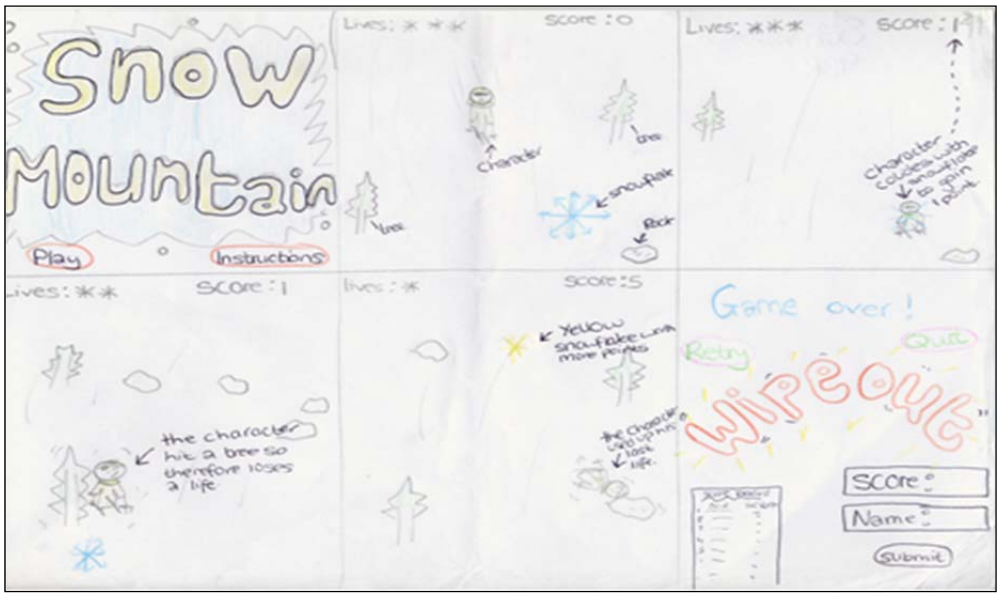
LWGW's storyboard



MH's storyboard



OW's storyboard



SW's storyboard