Computers & Education 80 (2015) 224–240

Contents lists available at ScienceDirect

Computers & Education

journal homepage: www.elsevier.com/locate/compedu

Learning to communicate computationally with Flip: A bi-modal programming language for game creation

Kate Howland*, Judith Good

The University of Sussex, Department of Informatics, Falmer, Brighton BN1 9QJ, UK

ARTICLE INFO

Article history:
Received 11 October 2013
Received in revised form
22 August 2014
Accepted 23 August 2014
Available online 16 September 2014

Keywords:
Evaluation of CAL systems
Interactive learning environments
Programming and programming languages
Secondary education

ABSTRACT

Teaching basic computational concepts and skills to school children is currently a curricular focus in many countries. Running parallel to this trend are advances in programming environments and teaching methods which aim to make computer science more accessible, and more motivating. In this paper, we describe the design and evaluation of Flip, a programming language that aims to help 11–15 year olds develop computational skills through creating their own 3D role-playing games. Flip has two main components: 1) a visual language (based on an interlocking blocks design common to many current visual languages), and 2) a dynamically updating natural language version of the script under creation. This programming-language/natural-language pairing is a unique feature of Flip, designed to allow learners to draw upon their familiarity with natural language to "decode the code". Flip aims to support young people in developing an understanding of computational concepts as well as the skills to use and communicate these concepts effectively. This paper investigates the extent to which Flip can be used by young people to create working scripts, and examines improvements in their expression of computational rules and concepts after using the tool. We provide an overview of the design and implementation of Flip before describing an evaluation study carried out with 12–13 year olds in a naturalistic setting. Over the course of 8 weeks, the majority of students were able to use Flip to write small programs to bring about interactive behaviours in the games they created. Furthermore, there was a significant improvement in their computational communication after using Flip (as measured by a pre/post-test). An additional finding was that girls wrote more, and more complex, scripts than did boys, and there was a trend for girls to show greater learning gains relative to the boys.

1. Introduction

Learning to think 'computationally' has long been recognised as important (Papert, 1980), and the recent computational thinking drive has refocused attention on this as a significant issue in modern computing (Grover & Pea, 2013a; Wing, 2006). As computation has become pervasive, underpinning communication, science, culture and business in our society, the ability to be a creator rather than just a consumer of technology is increasingly seen as an essential skill in order to participate fully in a digital society (McLoughlin & Lee, 2008; Resnick et al., 2009). The recent decision to introduce computer science teaching from primary school onwards in the UK reflects the growing recognition of its importance (Brown et al., 2013).

Although there is broad agreement that it is important to teach computational thinking skills from a young age, and to people who may never learn to program using commercial languages (Fletcher & Lu, 2009; Guzdial, 2008), deciding which specific skills should be taught is still an emerging endeavour. Perhaps the most fundamental of the evolving set of computational thinking skills is the ability to define clear, specific and unambiguous instructions for carrying out a process: in addition to being an important precursor for computer programming, it is applicable to many other domains of endeavour (Howland, Good, & Nicholson, 2009).

Specifying computational rules requires an understanding of basic computational concepts as well as the ability to use and communicate these concepts effectively. If we are to teach this ability to a wide range of young people, we need tools that make the activity motivating and accessible to children of all abilities, and to both girls and boys. Issues such as teacher training and support also play key roles (Robertson,

* Corresponding author. Tel.: +44 (0)1273 877218.
E-mail addresses: k.l.howland@sussex.ac.uk (K. Howland), J.Good@sussex.ac.uk (J. Good).

http://dx.doi.org/10.1016/j.compedu.2014.08.014
0360-1315/

Macvean, & Howland, 2013), but are beyond the scope of this paper. Although there are some encouraging low-tech approaches to teaching computer science and computational thinking skills in a school context (Bell, Witten, & Fellows, 2006), specially designed novice languages are usually required for young people with no prior programming experience to develop these skills.

Flip, the programming language introduced and examined in this paper, aims to help young people develop computational skills by scaffolding them as they script events while creating their own narrative-based computer games. A combination of two factors makes Flip distinct from other programming languages aimed at teaching young people computing skills. First, as a bi-modal language, it provides a plain English translation of programs composed using graphical blocks. This design feature is intended to help learners understand the meaning of the program beyond the specific arrangement of blocks. Second, it works in conjunction with a commercial game-creation environment which allows children to make substantial progress with their game design before facing the challenge of programming. Programming is introduced as part of the broader activity of game creation, which involves storytelling and world-design. Flip is designed to appeal to girls and boys with a range of interests, and to have a "low floor" (Papert, 1980), in other words, to be sufficiently easy to use such that all users can create simple scripts.

In this paper we give an overview of Flip, and present an evaluation of the language in a naturalistic classroom setting carried out over 8 weeks. Whilst studies of other novice languages (outlined in the following section) either rely on an analysis of the games created to evaluate learning, or use pre/post-tests in a format closely allied with the language of choice, our evaluation uses pre and post-tests written in natural language and designed to measure broader improvements in the understanding and expression of computation.

More specifically, the research questions addressed in this paper are:

1. Will young people's use of Flip during an extended game creation activity improve their understanding and expression of key computational concepts (as evidenced by their ability to write computational rules in natural language)?
2. Are any gender differences observed with respect to computational understanding and expression before and after using Flip?
3. Is improved understanding and expression demonstrated across all targeted computational concepts?
4. What types of computational errors do young people exhibit prior to using Flip, and does the use of Flip lead to a reduction in specific types of errors?
5. Does Flip enable young people with no prior programming experience to create their own working scripts?
6. Are there any gender differences in terms of the scripts created using Flip?
7. What is the effect, from a teacher's perspective, of using Flip in the context of a game creation activity?

The following section presents a literature review of relevant work on the teaching and learning of computational skills, visual language design and evaluation, and gender issues. In Section 3, the Flip language is described, including an overview of the background and design process. In Section 4 we outline the design and method for the evaluation study. Section 5 presents the results of the evaluation study, which are then discussed in Section 6 before conclusions are drawn.

2. Literature review

2.1. Computational thinking

Computational thinking has been much discussed since Wing introduced the term, defining it as "solving problems, designing systems and understanding human behaviour, by drawing on the concepts fundamental to computer science" (Wing, 2006, p. 33). Wing's 'call to arms' article made a huge impact, but provided little in the way of details about what constituted computational thinking, how it could be taught or how knowledge of computational thinking could be evaluated. At heart is the idea that computer science is not just about programming but in fact encompasses a broad range of useful and interesting ways of thinking, and involves a surprising amount of innovation (Denning & McGettrick, 2005).

The importance of teaching computational thinking skills to everyone from a young age is a key element of the concept, and one which has captured the attention of many researchers. For Guzdial (2008) this presents an important challenge for designers of programming languages aimed at novices, whilst Fletcher and Lu (2009) believe that computational thinking should be taught far in advance of contact with programming languages. The Computer Science Unplugged initiative aims to teach young people computing skills and concepts (such as binary numbers and search algorithms) without using a computer, instead encouraging them to take on the role of the computer themselves (Bell, Alexander, Freeman, & Grimley, 2008).

Although agreement has not yet been reached on a computational thinking curriculum, abstraction is held to be a particularly important high-level concept (Kramer, 2007; Nicholson, Good, & Howland, 2009; Wing, 2008). The communication of computational concepts is also held to be particularly important, with Cortina (2007) including the use and expression of algorithms as key components of a computational thinking curriculum. Similarly, in the context of primary and secondary curricula for computation, Lu and Fletcher (2009) stress the importance of establishing vocabularies and symbols which can be used to introduce and explain computation and abstraction. They argue that without the use of an appropriate form of notation, it is hard to develop the appropriate mental models (Lu & Fletcher, 2009). The specific area of computational communication we aim to develop through Flip is the ability to define clear, specific and unambiguous instructions for carrying out a process (Howland et al., 2009).

2.2. Visual languages

The use of graphical code building blocks has shown considerable promise in languages which aim to give novices their first introduction to computation. Scratch, one such language, allows users to drag and drop graphical blocks to compose simple programs which, in turn, allow them to create simple games (Maloney, Kafai, Resnick, & Rusk, 2008). Scratch is currently very popular in schools in the UK, and there is evidence that it can help children improve their understanding of a subset of computer science concepts based on pre and post-test data (Meerbaum-Salant, Armoni, & Ben-Ari, 2010). Although Meerbaum-Salant et al.'s findings are encouraging, many elements of the post-test

were closely tied to the Scratch programming language, making it hard to distinguish pupils' understanding of the Scratch from their understanding of computational concepts more generally.

Alice is a novice programming environment in which scripts are composed using blocks with snippets of pseudo code (Cooper, Dann, & Pausch, 2000; Dann & Cooper, 2009). Alice is most often used with college-aged students, where there has been evidence of beneficial learning outcomes, particularly for students with minimal programming experience or a poor mathematical background (Moskal, Lurie, & Cooper, 2004; Sykes, 2007). The language has also been used with middle-school children, most notably in the form of a specially designed storytelling version of the language (Kelleher, Pausch, & Kiesler, 2007). Within-language post-tests indicated a good understanding of computational concepts by girls who used Storytelling Alice,[1] but there was no pre-test to allow measurement of learning gain.

Agentsheets (Repenning, Ioannidou, & Zola, 2000) is an environment which allows novice end-users to quickly create simple games (based on classic games such as Frogger). A recent study investigated the extent to which teachers and college students could recognise computational thinking patterns after creating games with Agentsheets (Basawapatna, Koh, Repenning, Webb, & Marshall, 2011). This study involved a post-test 'quiz' where concepts are applied in real-world scenarios, distinct from Agentsheets. The results of this ingenious quiz suggest good transfer, although the findings may not apply to younger users, and there is no pre-test to allow measurement of learning gain. A study with middle-school children using Agentsheets did not use the quiz to measure transfer, and instead focussed on the computational thinking patterns demonstrated in the games created (Basawapatna, Repenning, & Lewis, 2013).

Measuring the learning of computational concepts by examining the games created (instead of setting separate tests) has been used to indicate the promise of other languages. A study of games created by middle-school girls using Stagecast Creator suggests that they demonstrate an understanding of computer science concepts (Denner, Werner, & Ortiz, 2012). The thorough analysis of these games provides good evidence that young people can work with computer science concepts when creating games, but leaves open the question of transfer. Similarly, an analysis of games created using ScriptEase, a GUI-based nested scripting system, found that abstraction and higher-order thinking skills were demonstrated within the games created, however, learning outcomes were not measured separately (Carbonaro, Szafron, Cutumisu, & Schaeffer, 2010).

Other languages considered to have potential for teaching computational concepts in a game design context include Kodu (Stolee & Fristoe, 2011; Touretzky, Marghitu, Ludi, Bernstein, & Ni, 2013), Lego NXT-G (Touretzky et al., 2013), the Mission Maker 'Rule Editor' (Immersive Education, 2007) and Android App Inventor (Grover & Pea, 2013b), however the studies focussing on these languages have not considered learning outcomes directly. For an analysis of the characteristics of novice languages which can support the development of computational thinking skills, see (Howland et al., 2009).

There is good evidence that many of the existing languages discussed above can offer a way for novice programmers to engage in coding tasks, but they have not been designed to encourage the development of a more general understanding of computational concepts. Through its novel natural language mode Flip aims to address the issue of transfer, and to help young programmers develop a language of computation which aids their understanding.

2.3. Gender issues

There are persistent concerns about the underrepresentation of women in computing fields (Klawe, Whitney, & Simard, 2009), particularly in light of the encouraging elimination of the gender gap in maths-related subjects at school level (Cheryan, 2012). It has been argued that 'transformational interventions' are required to give young women the chance to develop identities as experts in computing (Soe & Yakura, 2008). Software tools are generally intended to be gender agnostic, and there are relatively few examples of research which compares the usage of software by males and females. Where these issues have been investigated for problem-solving software, there is evidence of significant differences in feature usage and willingness to explore features between males and females (Burnett et al., 2011).

Accordingly, it is important to ensure that new tools developed to support the learning of computation from a young age are motivating, accessible and effective for girls as well as boys. Some evaluations of novice game design languages have considered the issue of gender by looking at girl-only groups. Kelleher et al. (2007) found that girls successfully learned basic programming constructs with Storytelling Alice, and were more motivated to program using this tool than the generic version of Alice, suggesting that storytelling holds appeal for girls in this context. Denner et al. (2012) found that girls engaged in moderate levels of complex programming activity when creating games using Stagecast Creator.

Other researchers have compared boys' and girls' attitudes to and performance in game creation tasks in mixed groups. Carbonaro et al. (2010) suggest that game creation offers a gender-neutral approach to teaching computer science. They found that girls' games created using ScriptEase exhibited greater numbers of higher-order thinking skills than boys, and that both genders were equally motivated by the activity. In a small study using Scratch, Baytak and Land (2011) found that girls create more scripts and used more statements overall than boys, although they note that sometimes functions can be created more efficiently with fewer commands. In a study of games created using the Adventure Author software, which includes some basic menu-based scripting, Robertson (2012) found that girls' games were rated more highly than boys' games, particularly on storytelling aspects. However, these positive findings in relation to performance may not translate to motivation. A further study by Robertson (2013), based on pre and post attitude questionnaires from 225 children, found that although girls enjoyed game making, they enjoyed it less than boys, and stated that they were less inclined to study computer science in the future after taking part in a game creation project.

3. The Flip language

3.1. Background

We have conducted extensive research on how young people can use commercial game creation software to develop their own 3D video games (Good, Howland, & Nicholson, 2010; Good & Robertson, 2006b; Howland, Good, & du Boulay, 2008, 2013). We have found this type of

[1] Now called Looking Glass: http://lookingglass.wustl.edu/.

game creation to be very motivating for young users, who are drawn to the activity for a variety of reasons, including personal interest in computer games, art and design activities or telling stories. Since each individual has the opportunity to create their own unique and highly personal game, the activity is learner-led and the game ideas are learner-generated. As a result, in the numerous game creation workshops we have facilitated, we have found that young people are able to clearly describe their story ideas and the events that they would like to occur in their games. From there, it is only a small step for young people to start thinking in computational terms about how to write programs to bring about these situations. This provides an ideal context for introducing pupils to the often difficult topic of programming, and the related computational skill of rule specification.

The computer games software we work with is called Neverwinter Nights 2 (NWN2), and was published by Atari in 2006. NWN2 is a computer role-playing game (RPG) in which players explore a large fantasy world and take part in a dramatic interactive story, with the players' choices determining how the plot progresses. The game is part of the Dungeons & Dragons franchise, a series of 'medieval fantasy' RPGs. Included in the NWN2 software package is the Electron Toolset (Fig. 1), a professional game-development environment which was used by the developers, Obsidian Entertainment, to build a large part of NWN2. Using an existing toolset gives access to an extensive set of resources, and a powerful game engine. Other commercial games creation toolkits available for use by amateur enthusiasts, such as Unity (Unity Technologies, n.d.) and the Unreal Development Kit (Epic Games, n.d.) are incredibly powerful, but have a high floor and are inaccessible to most children of the target age range, particularly those without existing technical skills. Conversely, with the Electron Toolset, around five minutes of instruction is enough to empower users to begin independently working on their own 3D landscapes, complete with characters and landscape items such as rivers, trees and buildings. At the same time, the complexities of the toolset are such that, once learners have created a basic game, they will typically spend many hours, days or even weeks improving on and expanding their game. In this way, the Electron Toolset provides a good balance of complexity and accessibility.

Unlike game-based programming environments such as Scratch (Maloney et al., 2008), Alice (Cooper et al., 2000) and AgentSheets (Repenning et al., 2000), including those designed more specifically to support storytelling (Kelleher et al., 2007), users do not need to engage with computer programming from the outset in order to create something meaningful and impressive. Instead, users are able to create an area, or setting for their game, populate that setting by choosing from a vast selection of characters, scenery and props, and create conversations for their characters, all without needing to write any code. As such, they are able to explore the possibilities of the toolset, and use it to build something which appeals to their own sensibilities and over which they feel ownership, approaching game building first and foremost as a creative task.

This initially shallow learning curve is invaluable for motivation, and gives users the confidence to persevere with the activity. Once they have spent time designing areas, and customising them with props and characters, designers will begin to think about the story that they

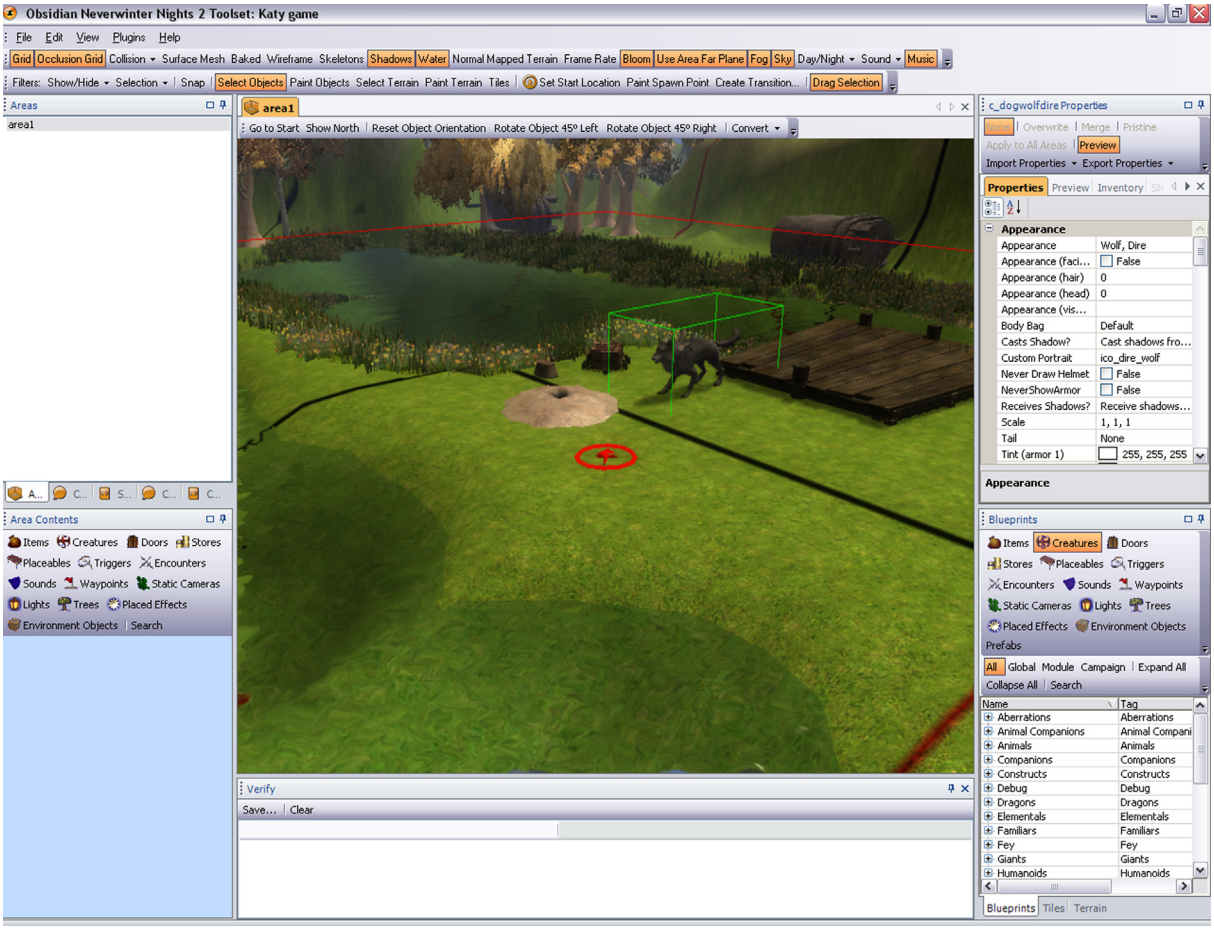


Fig. 1. Electron toolset.

would like the player to take part in within their game world. Given that the story will be interactive, a number of computational concepts can be introduced quite naturally to users, and they can begin to engage with programming. At this point, their investment in their game world and the narrative they are constructing drives them to try and achieve programming goals, and concepts such as correct rule specification, sequencing, tests and conditionals occur naturally. Because these concepts are introduced in the context of learner defined goals, they may be more meaningful and easier to grasp than when introduced in a more abstract manner (Boekaerts & Minnaert, 1999; Resnick, 1987).

Nonetheless, engaging with and mastering these concepts previously required use of NWScript, the Electron toolset's inbuilt text-based scripting language, which is similar to C in its syntax and level of complexity (see Fig. 2). Young users are almost invariably intimidated and frustrated by this element of the game creation process, and find it impossible to proceed without substantial help from experts.

Additionally, as described above, although we have found that young people can specify computational rules whilst describing narrative gameplay elements in the context of an informal conversation, they often require prompting from workshop staff before they can fully state the conditions and actions involved. In other words, they need support in translating their narrative view to a computationally complete view. This means that they face difficulties even before they hit the barrier of the intimidating syntax of NWScript. For this reason we developed Flip, to replace NWScript, and support young people in moving from their intuitive understanding of narrative events in gameplay towards a computational understanding.

3.2. Design of Flip

A Learner Centred Design (LCD) approach was adopted for the Flip language. We consider it important to design educational tools with extensive input from the target users from the outset, and we used the CARSS framework to guide our LCD activities (Good & Robertson, 2006a). In brief, CARSS allows one to plan for the learner centred, participatory design of educational environments with young people by considering: the *Context* in which the design activities will take place (including the inevitable constraints inherent in these contexts), the *Activities* that are appropriate at various points in the design cycle, the *Roles* that need to be fulfilled by the design team members, the broad range of *Stakeholders* to be included in the design, and the *Skills* needed by both the adult and child design team members (including the extent to which these skills can be fostered as part of the participatory design process).

Our participatory design process involved a number of phases, which are described in more detail in (Good & Howland, submitted). We started the participatory design phase with requirements gathering, conducting sessions in which we taught non programmers to use Inform 7, a natural-language based programming environment (Nelson, 2006) and subsequently asked them to use it to create short pieces of interactive fiction. This allowed us to gather data on the potential utility of natural language as a basis for the programming language we aimed to design. More specifically, we looked at the difficulties encountered by these non programmers as they engaged in tasks requiring them to both create and understand simple statements in the Inform 7 language. One of the primary findings of this work was in understanding the difficulties created by the inherent lack of specificity in natural language. For example, Inform 7 uses particular natural language words as keywords, or commands, e.g. “Instead”, which then allows users to write a rule such as “Instead of going through the door”. However, at the time of creating a rule, users may not remember if the keyword sequence should be “Instead of” or “Rather than”, both of which are functionally equivalent in natural language, but only one of which will produce the desired result. Essentially, Inform 7 is a subset of natural language, but it can be very difficult for the user to remember which specific subset it is. The results of this study suggested that although natural language may be helpful in *understanding* programs, it can create unnecessary complications when *writing* programs.

We then conducted a study to look specifically at the ways in which young people naturally describe game-based events which involve computational concepts (Good et al., 2010). We asked them to watch a walkthrough of a game in Neverwinter Nights 2. At various points, the game was paused, and they were asked to write a rule, in their own words, which described the behaviour they had just observed. In this case, we were looking for patterns in young people's expressions that would guide us in our language design. In brief, we found that rules were primarily written in an event-based format. Furthermore, we found that approximately 80% of the statements contained errors of some type, in particular, omitting important information from the rule (rather than including erroneous information). This suggested that even when syntax is not an issue, the semantic aspects of rule specification still prove tricky for novice users. As a result of this study, we decided to implement an event-based language (which also fits with the interactive nature of programming for games), and to provide explicit support for the semantic aspects of rule specification.

Following this phase of requirements gathering, we moved to the design phase, asking young people to design their own visual representations for different rule elements. We also produced low-fidelity mock ups of the Flip language, and tested them to ensure that young people were able to use them both for comprehension and generation tasks. Finally, we ran a four-day workshop in which a full prototype of the Flip language was used by young people aged 11–15 so as to ensure that the language did not suffer from usability problems, and that users could carry out both comprehension and generation tasks.

By working with a wide range of young people to gather requirements, develop understandable representations and test prototype designs before implementing them, we were able to ensure that the tool could meet our goals.

Fig. 2. NWScript conditional example.

3.3. Flip language description

Flip features a simple visual interface in which users create scripts by connecting graphical 'blocks' together. However, Flip is also bi-modal: as users add graphical blocks to their program, Flip dynamically generates a full natural language equivalent of the script under construction. This allows users to check, at any point, the meaning of the program they are working on by comparing the natural language with the graphical blocks. The natural language description is updated every time a change is made, so users can check the effects of each individual block on the program as a whole, and trace the development and modification of their script at a micro level. Fig. 3 shows a labelled view of the Flip interface.

Fig. 4 shows a simple script in which the player is rewarded once they have slain the dragon. There are two action blocks attached to the pegs on the spine, one which displays a message to the player, and another which gives the player some gold coins. Actions are run in order from the top down: first the message is displayed, then the coins are given. The event block at the top in the event slot dictates when the script will happen: this script will run when the dragon Smaug is killed. Below the Flip script in Fig. 4 is the equivalent NWScript code that, when manually added to the 'On Death Script' field of Smaug's properties sheet, would create the same behaviour.

Fig. 5 shows examples of the different categories of Flip blocks which can be used to compose scripts. **Actions blocks** are used to make something happen in the game. Fig. 5 shows an example action block which will give a certain number of coins to a character in the game. The grey slots must be filled with **game blocks** or **value blocks**, and the text on the grey slots describes the type of block that can go there. **Game blocks** represent game objects, such as characters, doors and items, and **value blocks** are used to represent words and numbers. The value of word and number blocks is changed by clicking on the 'edit' button. **Event blocks** are used to determine when a script will be triggered. A script cannot be saved to the module until it has been given an event. The event shown in Fig. 5 needs to know which door (or placeable) must be unlocked in order for the script to run.

Condition blocks check whether a given condition is true. Users can make different things happen in their scripts depending on whether a condition is true or false. For example, the condition shown in Fig. 5 checks whether a creature is carrying a particular item. A young game designer might want to check whether the player has stolen a dragon's most prized possession, say, a magical amulet, in order to have the dragon act differently towards the player if they are indeed carrying the amulet. To make scripts check conditions in Flip, a **control block** is required.

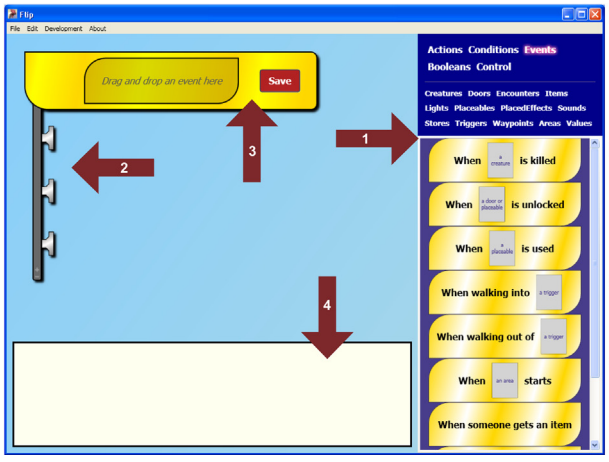
Control blocks allow users to make more complex scripts that react to the player's choices and the current state of the game world, and introduce computational concepts that are likely to be of wider use. Fig. 5 shows an 'If...Then' block. The light red slot (in the web version) in this block takes a condition, such as the one shown above it in the same figure. **Control blocks** also have their own spine. Any actions attached to this spine will only be triggered if the condition in the slot is true. Once the actions on the 'If...Then' block have been run, the game goes back to running the main script. That is, if there are more blocks underneath the 'If…Then' block in a script, they will run whether the condition placed in the 'If…Then' block is true or not.

Flip also has an 'If…Then…Else' **control block**, which has two spines. The actions on the first spine will run if the condition is true, while the actions on the second spine will run if the condition is false. Continuing our previous example, an 'If…Then…Else' block could be used to specify an alternative action sequence which should happen if the player is not carrying the amulet, as shown in Fig. 6. The natural language box for this script would say 'If the player is carrying Amulet of Power in its inventory, then Smaug attacks the player, otherwise a message pops up ("You escaped, but without the amulet.").'

Boolean blocks are used to make more complex checks on conditions. There are three types of **Boolean block**: the Or block, the And block and the Not block, as shown in Fig. 5. **Boolean blocks** can be used anywhere a standard **condition block** can be placed, and require the addition of one or more **condition blocks** so they can return a value of true or false that can be checked.

The Or block has slots for two conditions. Fig. 7 shows an Or block which has been given two conditions: 1) the player is carrying the amulet, and 2) the player has got 100 gold coins. It is true if either or both of its conditions are true, representing an inclusive Or. So the dragon will attack if the player has the amulet, or if the player has at least 100 gold coins, or both.

An And block only evaluates as true if both of its conditions are true, and the Not block is true only if the condition in its slot is not true.



1) The **Block Box** contains the blocks used to create scripts. Blocks of the currently selected category are displayed in the lower panel of the column.
2) The **Spine**, where the script is composed. The silver pegs are used to attach blocks to the spine. The blocks attached to the spine dictate what the script will do.
3) The **Event Slot**, where a single event block is placed to dictate when the script will execute.
4) The **Natural Language (or 'plain English') box**. As the script is built up, a natural language equivalent of the script automatically appears here.

Fig. 3. The Flip interface.

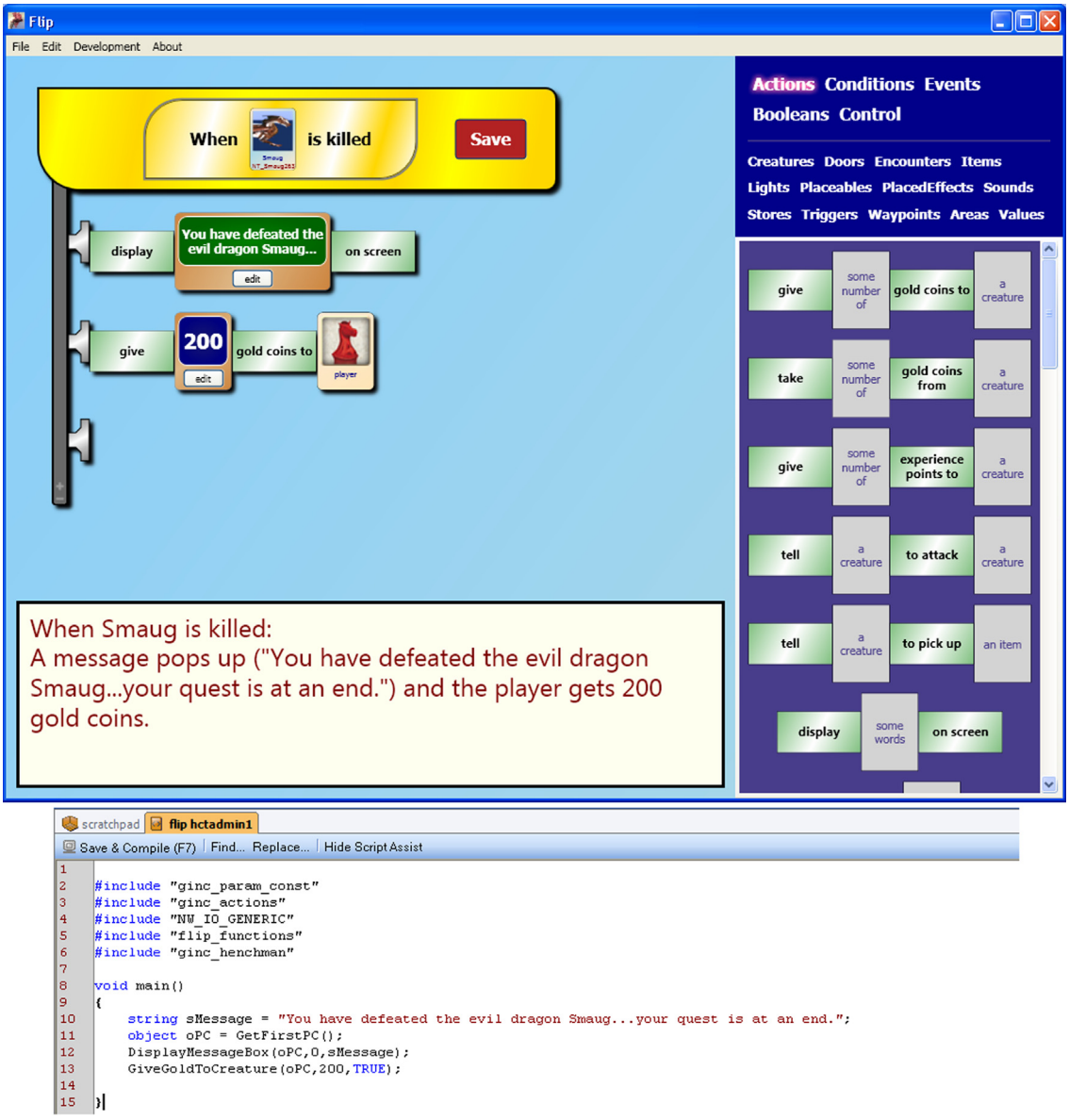


Fig. 4. A Flip script and the NWScript equivalent.

4. Study design and method

4.1. Study design

We designed a naturalistic evaluation to examine whether our target users were able to use Flip successfully, and to determine the extent to which their computational skills and understanding improved after using Flip. Given that the NWScript language is far too complex for most young people of the target age range to use, a comparative study between the languages was neither useful nor possible. Because Flip is designed to help young people improve their understanding and application of computational concepts, we focussed on measuring participants' ability to specify computational rules correctly, completely and unambiguously before and after using Flip. We chose a pre/post-test design with no control group. The tests were separate and distinct from the Flip language, to allow us to examine whether the understanding and skills developed could transfer beyond their learning of the programming language.

Rather than examining these issues in a lab-based setting, we chose to evaluate Flip in a real-world context in order to determine its effectiveness in the settings in which it would ultimately be used, i.e. in classrooms without the presence of researchers. Conducting the study in this way allowed Flip to be tested with a wide range of young people across classes, rather than just those who might volunteer for a university-based study. We provided training in using Flip for the class teacher in advance of the study, to ensure that the presence of researchers in the classroom could not influence results, either by providing more support to the teacher than he would normally have available, or by enthusing pupils through the novelty of new faces. The evaluation study was approved by the relevant university ethics committee, and informed consent was obtained for all participants.

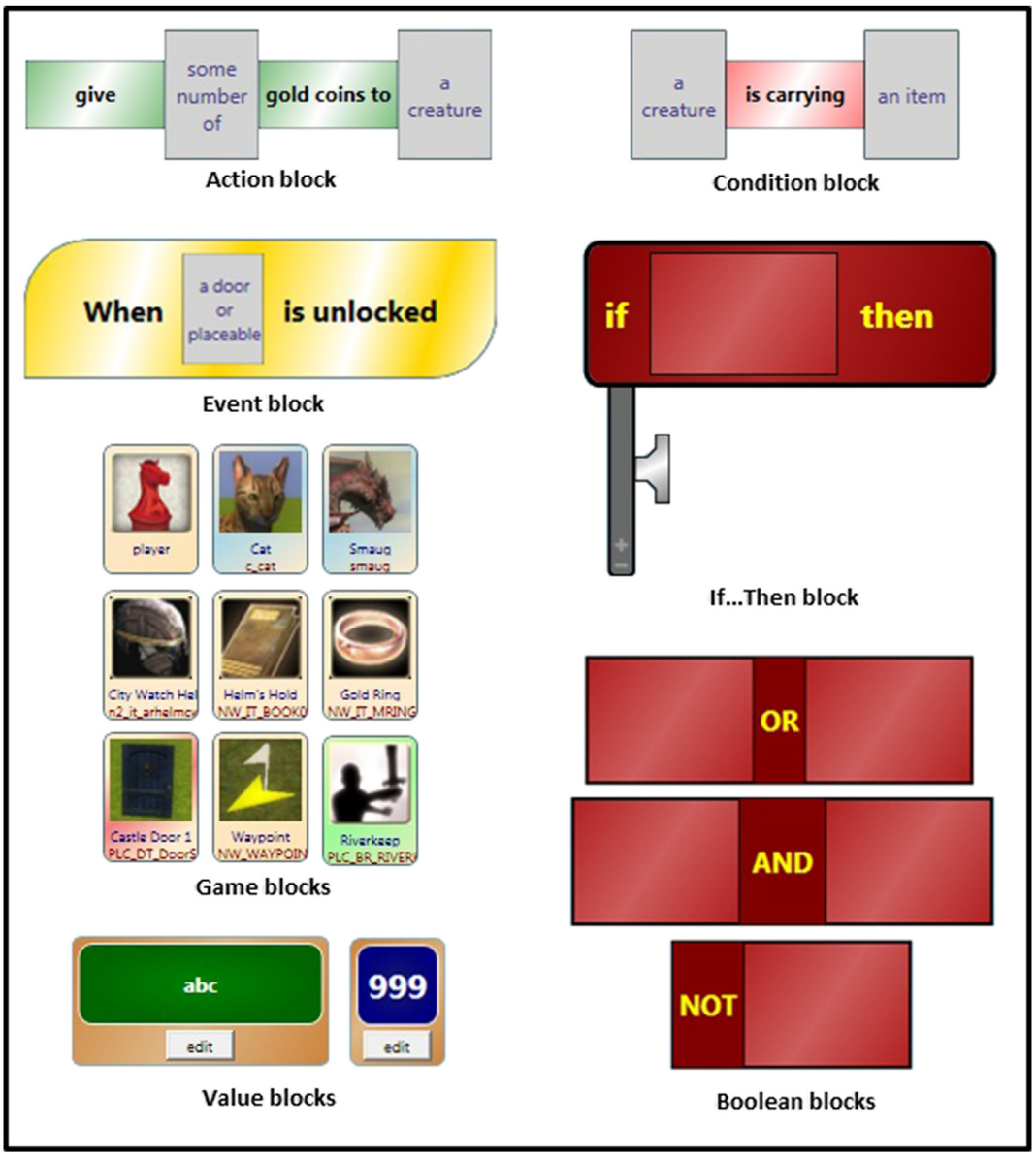


Fig. 5. Flip blocks.

4.2. Participants

Fifty-five young people aged 12–13 (29 girls, 26 boys) took part in a game creation project as part of their school ICT lessons. The pupils were spread across three classes in year S1 (the first year of secondary school) at a school in the East of Scotland, and all classes were taught the same material by the same ICT teacher. No pupils had previously been taught any programming in school. The school is a non-denominational, comprehensive secondary school, which generally achieves results in line with or slightly above the national average on standardised tests.

4.3. Materials

The materials used in this study were a matched pre and post-test, which investigated how young people expressed computational concepts when describing rules in games. The tests required participants to write computational rules explaining behaviours in a game, and were developed from previous work investigating the specification of computational rules by young people using natural language (Good et al., 2010). The tests used were piloted and refined through our past work, and have also been used to measure computational thinking in another large scale school based research project.[2]

[2] http://judyrobertson.typepad.com/adventure_author/research-materials.html.

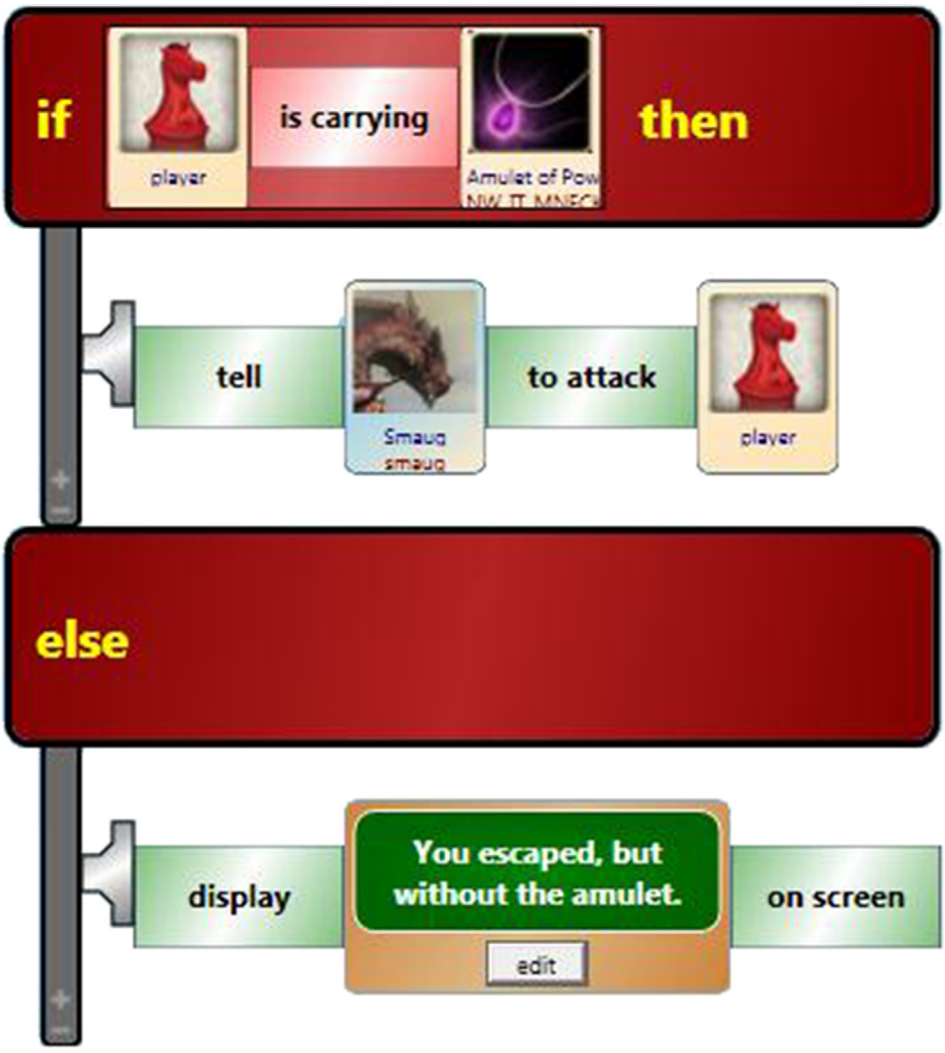


Fig. 6. Completed If...Then...Else block.

The pre and post tests were administered in conjunction with a video showing a playthrough of a game (one for each of the pre- and post-test) which was specially created for the study using the NWN2 Electron Toolset. Each game contained six scripted encounters which embodied particular types of computational structure, and increased in complexity.

The test items were administered in an online format. For each encounter shown in the game, students were asked to write a rule which could account for the behaviour they observed. The accompanying video automatically paused and showed the current question at the appropriate point.

Model answers were written to represent the rules which explained the events. These rules were broken down into rule segments, corresponding to functional components of the rule (e.g. the rule's condition, its outcome, etc.) Acceptable alternate phrasings were also written for each answer. Table 1 shows, for each of the encounters which required writing a rule, the question which the students were asked, the primary model rule describing the encounter, and classification of each of the functional components of the rule (e.g. condition, outcome).

Fig. 7. Completed If…Then block with Or block.

Table 1
Pre and post-test questions and model answers.[a]

Pre-test			Post-test		
Question	Model rule	Rule Segment	Question	Model rule	Rule Segment
Q1: What is the problem that Neeshka tells the player about?	Jeera the Sorceress has been causing havoc in the land (Optional: for example, burning down Neeshka's barn)	O	Q1: What is the problem that the village gatekeeper asks for help with?	The animals are acting crazy. (Optional: The village gatekeeper asks for help with them.)	O
Q2: Describe the two choices the player has when talking to Okku, and explain what happens in each case.	If the player answers "No way…"	C	Q2: Describe the two choices the player has when talking to the bear, and explain what happens in each case.	If the player answers "No…"	C
	then Okku walks away and sulks.	O		then the bear starts to cry.	O
	If the player answers "Yes of course I will help…"	C		If the player answers "Then I will set you free…"	C
	Then Okku gets unfrozen by a potion and joins the player on her quest.	O		The bear is freed and comes with the player.	O
Q3: Write a rule that explains how the well water affects different animals	If pigs or cows drink out of the well	C	Q3: Write a rule that describes how the bear reacts to different animals coming out of the portal	If wolves or deer come out of the portal	C
	they are fine.	O		then the bear lets them go.	O
	If the chickens drink out of the well	C		If skeletons come out of the portal	C
	they choke and die.	O		then the bear kills them.	O
Q4: Write a rule that describes how the pixie decides what to give the player.	If the player answers red	C	Q4: Write a rule that describes how the knight decides what to give the player.	If the player answers "true"	C
	he gives her a bloodstone.	O		then he gives her a book.	O
	If the player answers blue	C		If the player answers "false"	C
	he gives her a blue diamond.	O		then he gives her a sword.	O
Q5: Write a rule that describes the smiths' behaviour and what makes it change.	When the smiths are given the potion	T	Q5: Write a rule that describes the cats' behaviour and what makes it change.	When the player tells the big cat the enemies are gone	T
	they stop fighting and fall asleep.	O		they stop meowing and start purring.	O
Q6: Now, thinking back, write a rule that describes ALL the things that need to happen to get the smiths back to work.	When Okku shouts "boo"	T	Q6: Now, thinking back, write a rule that describes ALL the things that need to happen to get the cats out of the pit.	When the player plays the flute	T
	after the smiths have taken the potion and fallen asleep	C		after she has calmed the cats down	C
	they wake up and go back to work. (Optional)	0		they can jump out of the pit. (Optional)	0

[a] *Key for Rule Segments* T: Trigger, O: Outcome, C: Condition.

4.4. Procedure

The class teacher received training in using Flip at a half-day workshop organised by the authors, at which 10 secondary school teachers who had previously used a version of the NWN2 toolset with their classes were introduced to Flip and shown how to use it. Following the workshop, the teacher was provided with a manual and instructional videos which showed how various key tasks were carried out using Flip. Based on this training, the teacher planned a game creation project for the 3 classes to take part in over the course of 8 weeks.

Before the game creation project began, the pre-test was administered to all three classes. This required students to watch a video of a computer game being played: at regular intervals, they were asked to write a rule, in their own words, which would produce the behaviour they had just observed within the game. At the end of the game making project, a corresponding post-test was administered. Both tests had an identical number of questions, with each pre-test/post-test question matched in terms of level of difficulty, the specific computational concept being tested, and the rule structure(s) that would need to be present in an answer in order for it to be considered correct.

Each class had 2 lessons per week scheduled over the 8 week project, with each lesson lasting 53 min. However, bank holidays occurring during the 8 week period meant that one of the classes did not have the full 16 lessons on the project. The teacher introduced Flip to the pupils through whole class demonstrations, and gave examples of the different in-game effects that could be achieved using Flip.

Log files, and the scripts created during the project, were collected to allow investigation of the extent to which Flip was used by the pupils. The teacher was also interviewed at the end of the game making project.

4.5. Analysis

4.5.1. Pre and post-test analysis

The approach taken to pre and post-test analysis built on our earlier work looking at young people's descriptions of computational rules (Good et al., 2010). The scheme shown in Table 1 was developed to allow rules to be segmented into subsections according to the computational constructs they represent. Following this, for each rule section, we used an error coding scheme described in (Good et al., 2010) to look at correct and incorrect variants of the rule sections. The scheme distinguishes between *errors of omission* and *errors of commission*. An error of omission occurs when rule elements which should be present are left out, whereas errors of commission are due to the presence of elements which should not appear in the rule, because they are erroneous. The full error scheme is shown in Table 2.

The scheme was used to assign marks to the rules written by pupils: each correct rule segment (see Table 2 below) was given 1 point, while incorrect/incomplete rule segments were given 0 points. Furthermore, optional rule segments (namely, one optional rule segment in Question 6) were not allocated any points so as not to penalise students who chose not to include this additional, but not fundamental, information.

Table 2
Error analysis codes.

Category	Code	Meaning
Correct	C	All essential elements are expressed correctly and unambiguously
Errors of Omission	M	All elements of this section of the rule are missing
	PM	Some elements of this section of the rule are missing
	I	This section of the rule has been left unfinished
Errors of Commission	E	Rule section contains only erroneous information
	PE	Rule section contains some erroneous information
	V	Rule section contains information which is ambiguous or vague

The model answers were used as a guide for marking, with alternative phrasings which preserved the semantic meaning of the rule section also accepted. Similarly, we accepted any rule segment ordering, provided that differences in ordering did not change the semantic meaning of the rule. The overall requirement for marking a rule section as correct was that the event was described completely and unambiguously and the description included the key elements of the model answer.

Tables 3 and 4 shows how the marking scheme is applied. Table 3 shows the marking scheme for a question which asks pupils to write a simple rule containing a trigger behaviour, and an outcome. A model answer is given, along with notes to help the coder determine variations on the model which can be considered as acceptable.

All of the study data was coded by the first author, while a randomly selected sample of 20% of responses was coded by the second author. Inter-rater reliability was determined using Pearson's r to measure the correlation between the scores from the two raters, and Cohen's Kappa to measure the agreement between their error coding. For the pre-test there was a correlation of .978 ($p < 0.001$) on scores and a Kappa value of .893 ($p < 0.001$) on the codes. For the post-test, the scores correlation was .968 ($p < 0.001$) and the Kappa value was .864 ($p < 0.001$). These scores indicated high inter-rater reliability for the scores and high inter-rater agreement for the coding.

4.5.2. Script analysis

As well as the pre and post-test measures described above, we looked at the use of Flip over the game creation activity in the school sessions. The scripts created by each pupil were examined by the first author, recording the number of unique scripts created, the number of script saves that were logged (due to incremental changes to existing scripts) and the number and type of events, actions and conditionals used in the scripts.

5. Results

5.1. Pre and post-test measures of computational skills

5.1.1. Overall measures of computational skill

In order to measure any improvement in overall rule specification ability, we looked at the mean scores on the pre and post-tests. Both tests comprised six questions, and each test was worth 17 marks in total. Fig. 8a shows box plots of the pre and post-test scores. The mean score was 11.43 (SD = 3.44) on the pre-test and 13.25 (SD = 2.76) on the post-test, a difference which was highly significant on a paired samples t-test[3] ($n = 53$,[4] $t = -3.78$, $p < 0.001$).

In addition to looking at absolute differences in mean pre-test and post-test scores, we also calculated normalised learning change, which takes into account the maximum possible gain or loss given the pre-test score (Marx & Cummings, 2007). Normalised learning change is a variant on normalised learning gain which is appropriate for situations in which there are instances of negative learning gain (Knight, 2010), which was the case for a small number of the pupils. To calculate normalised learning change, learning gain is calculated as $100 \times (\text{post} - \text{pre})/(100 - \text{pre})$, and a modified calculation is used for pupils with negative learning gain: $100 \times (\text{post} - \text{pre}/\text{pre})$. Overall, there was a positive learning gain of 33.33%.

5.1.2. Measures of specific computational concepts

We were also interested in looking at changes in understanding across different computational concepts, namely, enhanced understanding of triggers, conditions and outcomes, in order to investigate whether the improvements were specific to certain computational concepts or occurred across all types. Two rule segments were categorised as triggers, seven as conditions, and eight as outcomes. Fig. 8c shows the percentage correct for each concept type on pre and post-tests. The difference between the pre and post scores was not significant for triggers ($n = 53$, $Z = -1.46$, n.s.), significant for conditions ($n = 53$, $Z = -2.08$, $p < 0.05$) and highly significant for outcomes ($n = 53$, $Z = -3.30$, $p = 0.001$) on Wilcoxon signed ranks tests.

5.1.3. Types of errors in rules

Finally, we looked at the types of errors in the pre and post-test answers. Overall, 33% of rule segments were incorrect on the pre-test, as compared to 20% on the post-test. Incorrect rule segments were coded in terms of the *type* of error manifested in the segment. Fig. 8b shows the percentages of incorrect answer segments by error type on the pre and post-tests.

[3] Throughout the results, where parametric tests are presented the data is normally distributed, with no significant outliers. Where these assumptions are not met, equivalent non-parametric tests are used.

[4] Although 55 pupils took part in the study, as one pupil was absent on the date of the pre-test and another was absent on the date of the post-test, the analysis was carried out on the data for the remaining 53 pupils.

Table 3
Sample model answer.

Question 5	Marks	Rule part	Model rule	Notes
Write a rule that describes the smiths' behaviour and what makes it change	1	Trigger 1	When the smiths are given the potion	Accept 'If'/'after' the smiths are given the potion or other equivalent keyword
	1	Outcome 1	they stop fighting and fall asleep.	Accept 'they fall asleep/they lay down'

Table 4
Sample pupil answers and marks.

Answer	Marks awarded	Analysis codes
Jeera put a spell on them and Abigail gave them a potion and they fell asleep !!!!! (how are they supposed to make weapons now?????)	2/2	C, C
they where triked by jeera and they are fighting and making a fool of them self.	0/2	M, M
the smiths were argueing because the sorceress has tooken away their trust for eachother and they stop this when they take a potion that makes them sleep.	2/2	C, C
tricked by jeera to hate each other and one says the other cheated but when they got the potion they are fine	1/2	C, V

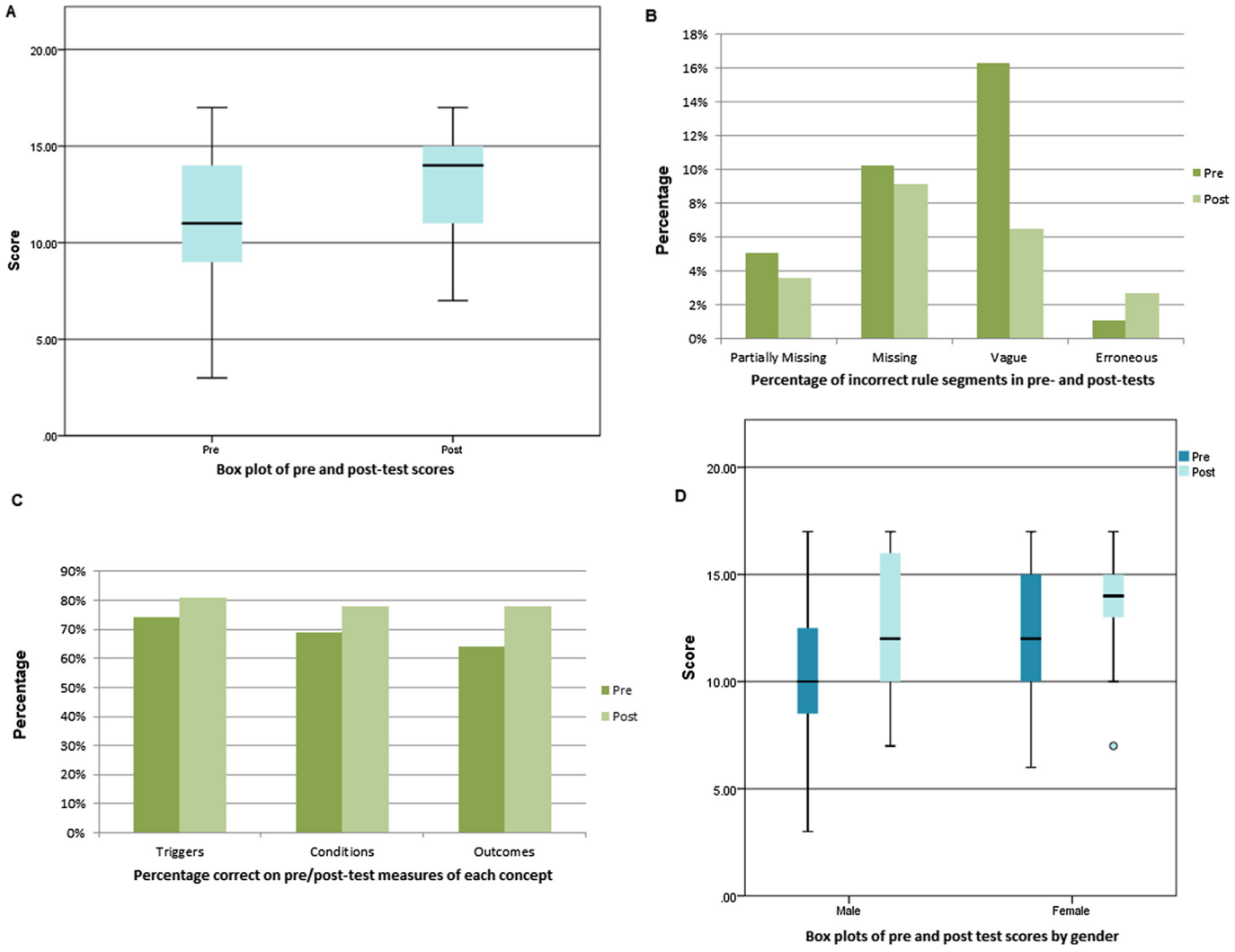


Fig. 8. Pre and post-test measures.

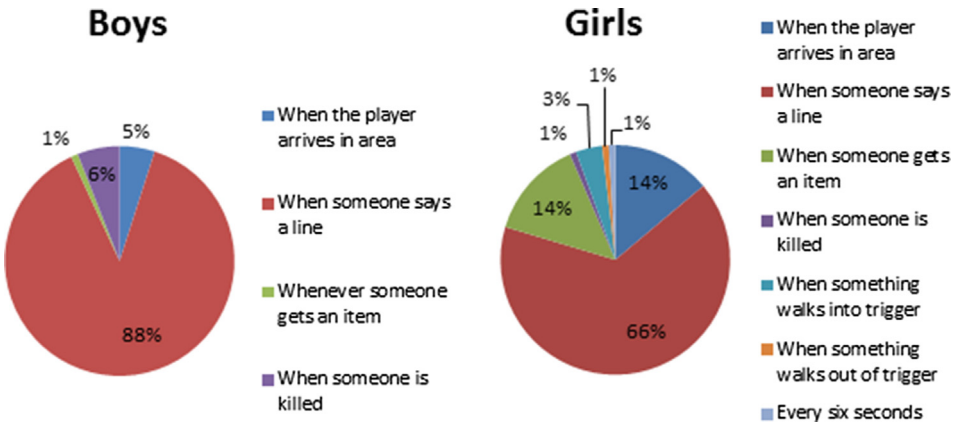


Fig. 9. Triggers used by pupils in scripts, by gender.

Wilcoxon signed ranks tests were carried out to examine whether the distribution of error codes changed significantly from pre to post-test. There were fewer missing and partially missing rule segments in the post-test as compared to the pre-test, however, neither of these differences were significant. There was a highly significant difference in terms of vague rules ($n = 53, Z = -3.87, p < 0.001$), with fewer vague rule segments in the post-test. Finally, and interestingly, there was a significant increase in the number of erroneous rule sections in the post-test ($n = 53, Z = -2.30, p < 0.05$).

5.1.4. Pre-test/post-test results by gender

We then looked at gender differences with respect to the pre/post-test measures reported above. In terms of overall measures of computational skill, for boys the mean pre-test score was 10.46 (SD = 3.48) while the mean post-test score was 12.54 (SD = 3.15), a difference which was highly significant ($n = 26, Z = -2.61, p < 0.01$). For girls, the mean pre-test score was 12.37 (SD = 3.20) while the mean post-test score was 13.93 (SD = 2.16), a difference which was significant ($n = 27, Z = -2.02, p < 0.05$). Fig. 8d shows the scores by gender.

Given that the girls' scores were higher overall on both the pre- and post-test, we also calculated learning change for both genders, as described in Section 5.1.1. As the girls' scores were significantly higher than the boys' on the pre-test ($n = 54, t = -2.10, p < 0.05$) there was less room for potential improvement, so taking into account the maximum possible gain or loss given the pre-test score was particularly important here. Learning change for both genders was positive (i.e. both genders exhibited learning gain): the mean learning gain for boys was 31.30%, while for girls it was 35.30%. This difference was not significant.

We also considered differences in error patterns by gender: these patterns largely mirrored the overall error pattern (shown in Fig. 8c), with the only significant difference being a reduction in vague rule segments for both boys ($n = 26, t = 3.25, p < 0.01$), and girls ($n = 27, t = 3.039, p < 0.01$).

5.2. Analysis of pupil created Flip scripts

5.2.1. Results overall

The scripts which were written by the pupils during the course of the game creation activity were logged and analysed.

Of the 55 pupils who took part in the game creation project, 43 (78%) managed to successfully create a working script for their game, complete with a triggering event and one or more actions. Overall, 411 complete and correct scripts were written and saved, however, these saves included revisions to already created scripts. Considering distinct individual scripts alone, a total of 210 scripts were created. This equates to a mean of 3.89 scripts per pupil overall, or 4.88 per pupil for those who created scripts.

In looking at the scripts created, we examined the types of computational constructs present in each script. At its most basic, a script will contain, after the trigger event, a single action. More complex scripts will contain more than one action, appropriately sequenced, while further complexity is evidenced by the inclusion of conditionals (either simple "If…Then" conditionals, or more complex "If…Then…Else" conditionals), and finally, by the inclusion of Boolean operators within the conditionals. All of the 42 pupils who created scripts created a simple script (trigger + action). In addition, 31 pupils (74%) created one or more complex scripts, i.e. their scripts contained additional constructs beyond the basic requirements for a well-formed script. 30 pupils (71%) created a sequence of two or more actions.

In terms of conditionals, seven pupils (17%) included a conditional within their script by using an If…Then…command or an If…Then…Else command to create a more complex logic within their scripts. In total, the seven pupils implemented 14 conditionals in their scripts, a mean of 2 per pupil.

5.2.2. Results by gender

Twenty-four girls (86%) created scripts compared to 18 boys (69%), an association that was not significant ($\chi^2(1) = 2.12$, n.s.) Of the 210 scripts created, 102 were created by boys, and 108 by girls.

A range of different events were used by pupils in order to trigger their scripts. Fig. 9 shows these events, split by gender. Overall, girls used a greater number of different triggers (7) compared to boys (4). For both genders, the most commonly used trigger was the speaking of a conversation line. Scripts triggered by conversation lines are easiest to implement, and the ones that pupils will typically learn first. Other triggers, such as scripting an action to take place when a player arrives in a specific area, require more skill.

Boys used the conversation line trigger 88% of the time, followed by 'when someone is killed' (6%), 'when the player arrives in an area' (5%), and 'when someone gets an item' (1%). In contrast, although the speaking of a conversation line was also the most frequently used

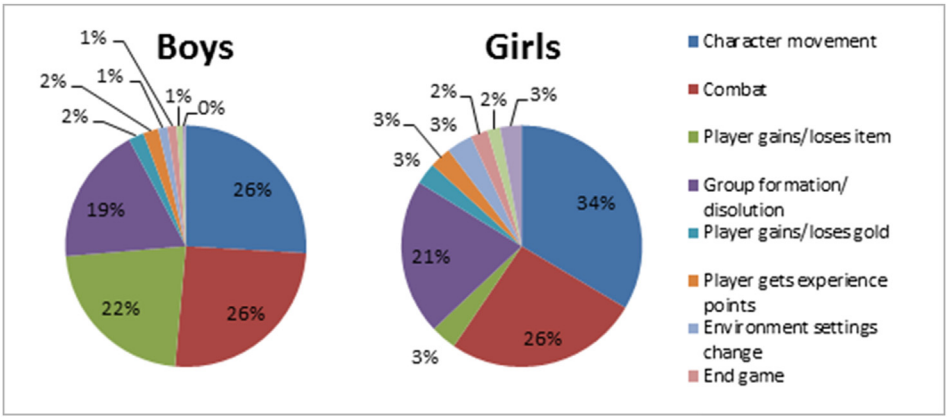


Fig. 10. Action types used by pupils in scripts.

trigger by girls, it accounted for 66% of the triggers used (as compared to 88% for the boys). They then used 'when someone gets an item' and 'when the player arrives in an area' with equal frequency (14%), followed by four other less frequently used triggers (as shown in Fig. 9).

A range of different types of actions were included in pupils' scripts. These actions follow on from the triggers shown in Fig. 9. Fig. 10 shows the number and type of actions which pupils used in their scripts, split by gender. The usage of action types was broadly similar between boys and girls, although girls used more character movement actions, and boys used more gaining and losing item scripts.

Of the 24 girls creating a script, 17 created a sequence of actions (i.e. the scripts they wrote contained more than one action), while of the 18 boys who created scripts, 13 created a sequence. In total, 413 actions were programmed by pupils (170 by girls, 243 by boys), with the mean number of actions per script being 1.97.

Of the seven pupils who used conditionals in their scripts, 5 were girls and 2 were boys. Fig. 11 shows the range of conditional commands used by pupils.

5.3. Teacher interview

At the end of the school term in which the game making sessions took place, a follow up interview was arranged with the class teacher via Skype. The interview was semi-structured, and focussed on understanding how Flip was introduced to pupils, the length of time it was used, and the extent to which various programming constructs (such as conditionals) were mastered by pupils. The teacher was asked to compare his experience of using Flip with previous NWN2 game creation projects run without Flip. The interview also covered issues of usability, looking at the utility of Flip as a whole, as well as individual features such as the "plain English" box. Finally, we asked about any difficulties which either the teacher or pupils encountered, and ways in which Flip could be improved.

Overall, the class teacher felt that the use of Flip in conjunction with the game creation software he had used with previous classes offered a number of advantages over the game creation software on its own. Firstly, he felt that giving pupils the ability to write their own scripts allowed them to create games that were more complex generally, but that also had more well developed story lines:

> So being able to sort of push things a little bit further and the kids have been able to actually do things a lot more interesting and actually really pushing the possibilities of stories much more. I've actually found using Flip a really great addition and a great aid for actually developing the games and developing the stories.

Furthermore, he felt that the ability to create more complex games with Flip had a positive impact on pupil motivation as compared to the previous year, engaging a greater number of pupils, and for a longer period of time. He stated that engagement had been maintained

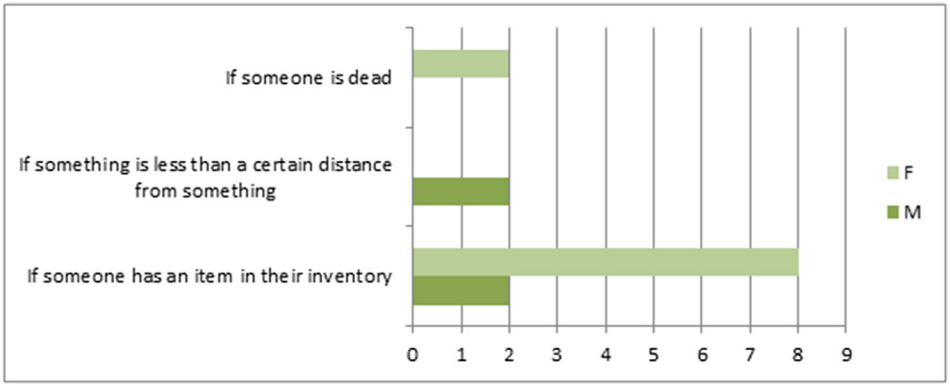


Fig. 11. Conditionals used by pupils in scripts.

throughout the term, with some pupils coming in at lunch breaks to work on their games, and continuing to come to the computer lab to put finishing touches on their work even after the course had ended.

In terms of introducing Flip, the teacher stated that he presented it not as a way of learning to program, but as a way for students to improve on their games.

> So I introduced it basically on the fly, at which point they then started to use the interface in developing the results for conversations. I also then started to develop it along the lines of 'enhancement of story'. For example, when an area was to begin, being able to have a character immediately run towards the person, signifying that maybe you should speak to this person and sort of try to develop the story like that. […], Again, showing how, if we use the programming side of it, we can then make our games even more complex, like the ability to unlock doors. […] But trying as much as possible to just treat it as a tool rather than specifically considering the programming side of it.

Although pupils may have not been aware that they were learning how to program, he nonetheless felt that they were beginning to master basic computational skills. In terms of specific skills, the class teacher felt that most pupils had been able to master the creation of simple scripts which could be triggered by conversations and other events, while some were able to move on to conditionals.

> [I]t allows a very clear and simple way to effectively introduce the idea of conditional statements and cause and effect in programming without them knowing. The whole idea of having structures and having dependencies and having results dependent on certain circumstances was fantastic. So it was really useful to be able to do these more technical aspects of creating a game without actually them realising they're programming and scripting, but also them developing the language and confidence to say, 'Oh, if I do this, then I'll have to do that' and them actually using the terminology without thinking about it.

The above excerpt highlights one of the most interesting qualitative findings from the project, namely, that using Flip, and specifically, the natural language output (or "plain English box" as it was referred to), allowed pupils to better understand the scripts they created, and supported them as they worked collaboratively on tasks such as debugging:

> [T]here were a lot of kids talking about it [the output of the plain English box], but the times that stick in my mind were when things weren't working. It was quite often when kids were trying to solve problems or they were helping each other out. They looked and they read what that was saying or would speak, "Oh look, look what you're doing". And quite often, the kid helping the other kid would actually look at the English box first, as if they were trying to figure out what they were wanting to do and then looking up to see what they'd done.

Although the teacher acknowledged that graphical programming languages are designed to be user friendly, he nonetheless felt that a natural language equivalent may boost learners' confidence in their scripting ability:

> There still may be the underlying lack of confidence in creating the programming structure, even though it's blocks, and using the English as that confirmation to say, "Yes, that is what I want to see".

In seeing their scripts as natural language equivalents, the teacher felt that pupils had been able to gain skills in using language in a computational manner, stating that they had learned about:

> The linguistic structures relating to decision trees and things like that, undoubtedly. I think generally just the comfort with using some technical terms and language, because I was able to direct people to set menus and things without really having to describe it. They were using the terminology … just saying, "Oh if you do this, then you have to do that". It was actually a greater confidence in using the language, so obviously having that is better.

6. Discussion and conclusions

In designing Flip, we aimed to create a graphical programming language that could be used within a game creation context and would allow learners to begin to take their first steps in computation. Given that young people are typically very motivated to produce their own game, we wanted to use this existing motivation to introduce programming as a means to an end, in other words, as a way of allowing them to progress even further in an activity they enjoy. Additionally, the games created with the Neverwinter Nights 2 toolset are heavily narrative based, and young people generate a number of very creative and elaborate story ideas which they can describe using natural language. Our aim in developing Flip was to give young people the tools to realise their many creative story ideas within the game, but also to build on their existing ability to express these ideas in natural language as a way of scaffolding their developing programming skills. By providing a natural language equivalent of their programs, young people were able to move between from a narrative view of their story ideas to a computational view, and then to move freely between these views.

Our interview with the class teacher suggests that we were able to achieve our broad aims, and that Flip allowed young people to create more complex games which embodied their gameplay and story ideas more fully as compared to the traditional programming language included with the game creation software. Furthermore, they were able to learn to use a number of computational constructs in their scripts. As part of this, they began to "learn the language of computation", talking about their programs with a greater degree of precision. The results of the pre- and post-test suggest that this skill is transferable to contexts where Flip is not being used, as there was a significant difference in the correctness of the young people's expression of rules after using Flip as compared to before. More specifically, the significant decrease in vague rules suggests that pupils are developing the necessary vocabulary to express rules in a more succinct and precise manner, that is less open to interpretation, a key aspect of learning to think computationally (Howland et al., 2009). This is positive, given

current movements to help all young people develop a basic set of computational skills (Brown et al., 2013; Wing, 2008). At the same time, we did observe an increase in the number of erroneous rule segments in the post-test, which is perplexing. In line with the decrease in missing and partially missing rules from pre to post test, it may be that pupils are making more attempts at writing complete and fully specified rules, which is positive. However, the cost of doing so may mean that, in some cases, these rules contain errors. This is similar to research in which students writing brief, high level summaries of computer programs made fewer errors than those who tried to describe the program's workings more fully (Good & Brna, 1998). More research would be needed to investigate this phenomenon.

Furthermore, our results were encouraging from a gender perspective. A substantial body of research exists which focuses on finding ways to motivate girls to engage with programming (Baytak & Land, 2011; Burge, Gannod, Doyle, & Davis, 2013; Carbonaro et al., 2010; Kelleher et al., 2007), but few directly compare relative performance across genders. In our study, we found that more girls created more scripts which were both more varied in terms of the range of actions they used, and more complex in terms of the computational constructs they contained. The fact that girls appear to be using more complex triggers is encouraging from a computational perspective. Furthermore, it suggests that girls are not relying solely on conversations to drive the plots of their stories, but are instead using events such as moving between areas or acquiring items as significant plot events.

In line with Kelleher et al.'s (2007) findings, it may be that embedding programming within a narrative-based activity makes it both more interesting, and more relevant to girls. Given that girls' attainment in literacy is higher than boys across all stages of the primary and secondary school curriculum (Education Standards Research Team, 2012), it may be that explicitly tying programming to an activity that they tend to do well in leads to a commensurate gain in their programming skills. In other words, if girls' stories are typically more complex and well developed, then when creating stories in games, their stories will also require more sophisticated scripts. This hypothesis would require further investigation, but would certainly be an interesting avenue to explore.

This issue also relates to a limitation of the current study, namely, that there was a significant difference between girls and boys in terms of their computational understanding on the pre-test. As none of the pupils had any prior programming experience, it may be that the design of the pre and post tests, which relied on pupils writing their answers in natural language, may have given the girls an advantage, given the differences in literacy attainment noted above. Using a multiple choice design would allow us to determine whether writing ability was creating a confound, but at the cost of a less complex and finely grained measure. Future work will include formal validation of the pre/post-tests, which will allow us to shed more light on this issue.

In terms of the scripts written and their complexity, the pupils' work was not as extensive as we might have hoped, with a relatively small number of conditionals used overall. Had we been involved in the field study on a day to day basis, it is likely that we would have encouraged further use of the language more explicitly, and might have seen concomitant increases in terms of number of scripts written, and their relative complexity. However, the positive aspect of the study was that it was naturalistic, which gives us good insight into how the language will actually be used by teachers 'in the wild', rather than into how the researchers wish it to be used (Robertson et al., 2013). Nonetheless, the teacher was very positive about its use, commenting that the introduction of Flip allowed pupils to begin to engage with computation in a way that had not been possible with the previous game making tool.

To conclude, we feel that there is benefit in activities which foster the development of computational skills for young people through activities that they typically enjoy, are motivated by, and, in some cases, for which they have an aptitude. Additionally, we feel that there is promise in developing environments which explicitly link young people's existing skills, in this case, an ability to describe narrative events in natural language, to skills to be developed, in this case, programming and computational skills more broadly. Further research will allow us to better understand and specify the nature of these links and how to provide optimum support in the environments we design.

Acknowledgements

Thank you to the teachers and pupils who so willingly took part in our participatory design work and evaluation studies. We are also very grateful to Keiron Nicholson for his work programming Flip. Finally, we would like to thank the anonymous reviewers who suggested many useful improvements for the paper. This research was supported by grant EP/G006989/1 from the Engineering and Physical Sciences Research Council.

References

Basawapatna, A., Koh, K. H., Repenning, A., Webb, D. C., & Marshall, K. S. (2011). Recognizing computational thinking patterns. In *Proceedings of the 42nd ACM technical symposium on Computer science education, Dallas, TX, USA* (pp. 245–250).

Basawapatna, A. R., Repenning, A., & Lewis, C. H. (2013). The simulation creation toolkit: an initial exploration into making programming accessible while preserving computational thinking. In *Proceeding of the 44th ACM technical symposium on Computer science education, Denver, Colorado, USA* (pp. 501–506).

Baytak, A., & Land, S. M. (2011). CASE STUDY: advancing elementary-school girls' programming through game design. *International Journal of Gender, Science and Technology, 3*(1).

Bell, T., Alexander, J., Freeman, I., & Grimley, M. (2008). Computer Science without computers: new outreach methods from old tricks. In *Proceedings of the 21st Annual Conference of the National Advisory Committee on Computing Qualifications (NACCQ08), Auckland, New Zealand*.

Bell, T., Witten, I., & Fellows, M. (2006). *Computer science unplugged*. from http://csunplugged.org/.

Boekaerts, M., & Minnaert, A. (1999). Self-regulation with respect to informal learning. *International Journal of Educational Research, 31*(6), 533–544.

Brown, N. C. C., Kölling, M., Crick, T., Peyton Jones, S., Humphreys, S., & Sentance, S. (2013). Bringing computer science back into schools: lessons from the UK. In *Proceeding of the 44th ACM technical symposium on Computer science education* (pp. 269–274).

Burge, J. E., Gannod, G. C., Doyle, M., & Davis, K. C. (2013). Girls on the go: a CS summer camp to attract and inspire female high school students. In *Proceeding of the 44th ACM technical symposium on Computer science education, Denver, Colorado, USA* (pp. 615–620).

Burnett, M. M., Beckwith, L., Wiedenbeck, S., Fleming, S. D., Cao, J., Park, T. H., et al. (2011). Gender pluralism in problem-solving software. *Interacting with Computers, 23*(5), 450–460.

Carbonaro, M., Szafron, D., Cutumisu, M., & Schaeffer, J. (2010). Computer-game construction: a gender-neutral attractor to Computing Science. *Computers and Education, 55*(3), 1098–1111.

Cheryan, S. (2012). Understanding the paradox in math-related fields: why do some gender gaps remain while others do not? *Sex Roles, 66*(3–4), 184–190.

Cooper, S., Dann, W., & Pausch, R. (2000). Alice: a 3-D tool for introductory programming concepts. *Journal of Computing Sciences in Colleges, 15*(5), 107–116.

Cortina, T. (2007). An introduction to computer science for non-majors using principles of computation. *ACM SIGCSE Bulletin, 39*(1), 222.

Dann, W., & Cooper, S. (2009). Education Alice 3: concrete to abstract. *Communications of the ACM, 52*(8), 27–29.

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 and Education, 58*(1), 240–249.
Denning, P., & McGettrick, A. (2005). Recentering computer science. *Communications of the ACM, 48*(11), 19.
Education Standards Research Team. (2012). *What is the research evidence on writing?*. Research report DFE-RR238. Retrieved accessed 12.07.13, from https://www.gov.uk/government/publications/what-is-the-research-evidence-on-writing.
Epic Games (n.d.). *Unreal Development Kit*, from http://udk.com/.
Fletcher, G., & Lu, J. (2009). Human computing skills: rethinking the K-12 experience. *Communications of the ACM – Association for Computing Machinery – CACM, 52*(2), 23–25.
Good, J., & Brna, P. (1998). Explaining programs: when talking to your mother can make you look smarter. In *Proceedings of the Tenth Annual Meeting of the Psychology of Programming Interest Group (PPIG-10)* (pp. 61–70).
Good, J., & Howland, K. (2014). Re-exploring the role of natural language in the design of novice programming languages: good for comprehension, bad for composition?. Submitted to the *Journal of Visual Languages and Computing* (submitted for publication).
Good, J., Howland, K., & Nicholson, K. (2010). Young people's descriptions of computational rules in role-playing games: an empirical study. In *2010 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*.
Good, J., & Robertson, J. (2006a). CARSS: 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 *Narrative and Interactive Learning Environments (NILE 06), Edinburgh, UK* (pp. 37–50).
Grover, S., & Pea, R. (2013a). Computational thinking in K–12 a review of the state of the field. *Educational Researcher, 42*(1), 38–43.
Grover, S., & Pea, R. (2013b). Using a discourse-intensive pedagogy and android's app inventor for introducing computational concepts to middle school students. In *Proceeding of the 44th ACM technical symposium on Computer science education, Denver, Colorado, USA* (pp. 723–728).
Guzdial, M. (2008). Paving the way for computational thinking. *Communications of the ACM – Association for Computing Machinery – CACM, 51*(8), 25–27.
Howland, K., Good, J., & du Boulay, B. (2008). A game creation tool which supports the development of writing skills: interface design considerations. In *Narrative and Interactive Learning Environments (NILE 08), Edinburgh, UK* (pp. 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. *Transactions on Edutainment, X*, 122–145. Springer.
Howland, K., Good, J., & Nicholson, K. (2009). Language-based support for computational thinking. In *2009 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), Corvallis, OR, USA* (pp. 147–150).
Immersive Education. (2007). *Mission maker*. Retrieved from http://www.immersiveeducation.eu/index.php/missionmakerm.
Kelleher, C., Pausch, R., & Kiesler, S. (2007). Storytelling alice motivates middle school girls to learn computer programming. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 1455–1464).
Klawe, M., Whitney, T., & Simard, C. (2009). Women in computing – take 2. *Communications of the ACM, 52*(2), 68–76. http://dx.doi.org/10.1145/1461928.1461947.
Knight, J. K. (2010). Biology concept assessment tools: design and use. *Microbiology, 5*.
Kramer, J. (2007). Is abstraction the key to computing? *Communications of the ACM, 50*(4), 36–42.
Lu, J., & Fletcher, G. (2009). Thinking about computational thinking. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education, Chattanooga, TN, USA*.
Maloney, J., Kafai, Y., Resnick, M., & Rusk, N. (2008). Programming by choice: urban youth learning programming with scratch. In *39th SIGCSE Technical Symposium on Computer Science Education, Portland, Oregon* (pp. 367–371).
Marx, J. D., & Cummings, K. (2007). Normalized change. *American Journal of Physics, 75*, 87.
McLoughlin, C., & Lee, M. J. (2008). Future learning landscapes: transforming pedagogy through social software. *Innovate: Journal of Online Education, 4*(5).
Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2010). Learning computer science concepts with scratch. In *Proceedings of the Sixth international workshop on Computing education research, Aarhus, Denmark* (pp. 69–76).
Moskal, B., Lurie, D., & Cooper, S. (2004). Evaluating the effectiveness of a new instructional approach. In *Proceedings of the 35th SIGCSE technical symposium on Computer science education, Norfolk, Virginia, USA* (pp. 75–79).
Nelson, G. (2006). *Natural language, semantics analysis and interactive fiction*. from http://www.informfiction.org/I7Downloads/Documents/WhitePaper.pdf Accessed 12.03.10.
Nicholson, K., Good, J., & Howland, K. (2009). Concrete thoughts on abstraction. In *Proceedings of Psychology of Programming Interest Group (PPIG 2009), Limerick, Ireland*.
Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York: Basic Books.
Repenning, A., Ioannidou, A., & Zola, J. (2000). AgentSheets: end-user programmable simulations. *Journal of Artificial Societies and Social Simulation, 3*(3).
Resnick, L. B. (1987). The 1987 presidential address: learning in school and out. *Educational Researcher, 16*(9), 13–20.
Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., et al. (2009). Scratch: programming for all. *Communications of the ACM, 52*(11), 60–67. http://dx.doi.org/10.1145/1592761.1592779.
Robertson, J. (2012). Making games in the classroom: benefits and gender concerns. *Computers and Education, 59*(2), 385–398. http://dx.doi.org/10.1016/j.compedu.2011.12.020.
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. http://dx.doi.org/10.1080/08993408.2013.774155.
Robertson, J., Macvean, A., & Howland, K. (2013). Robust evaluation for a maturing field: the train the teacher method. *International Journal of Child–Computer Interaction*, 50–60.
Soe, L., & Yakura, E. K. (2008). What's wrong with the pipeline? Assumptions about gender and culture in IT work. *Women's Studies, 37*(3), 176–201.
Stolee, K. T., & Fristoe, T. (2011). Expressing computer science concepts through Kodu game lab. In *Proceedings of the 42nd ACM technical symposium on Computer science education* (pp. 99–104).
Sykes, E. R. (2007). Determining the effectiveness of the 3D Alice programming environment at the computer science I level. *Journal of Educational Computing Research, 36*(2), 223–244.
Touretzky, D. S., Marghitu, D., Ludi, S., Bernstein, D., & Ni, L. (2013). Accelerating K-12 computational thinking using scaffolding, staging, and abstraction. In *Proceeding of the 44th ACM technical symposium on Computer science education, Denver, Colorado, USA* (pp. 609–614).
Unity Technologies (n.d.). *Unity 3D*, from http://www.unity3d.com.
Wing, J. (2006). Viewpoint-computational thinking. *Communications of the ACM – Association for Computing Machinery – CACM, 49*(3), 33–35.
Wing, J. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 366*(1881), 3717–3725.