1 Introduction

Digital technology is playing an increasingly important role in our lives. Software has become indispensable and pervasive in a relatively short period of time. Given the importance of software and digital products, there is a need for people who can develop these products, both at professional and end-user levels. In other words, there is a need for students undertaking Computer Science studies and careers but also for other professionals to be able not just to consume packaged software but to produce their own computational solutions to a wide variety of problems [1]. However admissions and retention rates to computer science university courses are falling [17], enrollment in computer science courses is male dominated [17] and although there is a thriving community of end-user programmers, there are serious concerns about the dependability of the software which they produce [1]. Therefore there is a need to both foster the development of computational thinking [19] in young learners and to motivate them to study computing subjects by improving the perception of computing, especially for girls.

One way of attempting to foster computational skills and provide motivation for computing subjects has been through environments designed to teach programming to children [10, 7]. It was thought that an understanding of programming would lead to improved academic performance in areas such as logical reasoning, problem solving and general maths skills. A further development consisted of creating authoring environments that enabled children to build sophisticated microworlds, simulations and computer games in an interactive way as an alternative to (but sometimes as well as through) programming [13, 14, 9]. Although there have been remarkable advances in these types of environments, young people still find some authoring tasks difficult to perform [5]. Additionally, some dangers of young learners spending too much time mastering programming skills in front of the computer (on top of the increasing amount of time they have to spend using the computer for other subjects) are problems related to Repetitive Strain Injuries [11] and lack of physical exercise due to passive routines.

We believe that three ways in which motivation and support of computing concepts for young learners can be improved are to design environments: for which programming is not an end in itself but a means to perform motivating and educationally relevant activities [12], that allow users to perform programming and scripting activities through more accessible ways and that employ digital technology that allows for a more active interaction in a physical sense. Specifically, we propose an approach to support the development of computational thinking skills in young learners through an environment that enables them: i) to undertake scripting activities as a way of building their own computer games, ii) to perform some of these scripting activities by acting and recording abstract scenes and iii) to employ embodied interaction [2] as the main user interface paradigm. These three characteristics are discussed below.

Authoring environments have enabled children to build sophisticated microworlds, simulations and computer games in an interactive way as an alternative to (but sometimes as well as through) programming [13, 14, 9]. However, frequently the emphasis in these environments is in programming. They are often presented as tools which will make learning programming easy, putting the content that can be created in second place. A recent approach, how-
ever, is to embed the programming activity within intrinsically motivating and educationally relevant activities, in other words, placing the content at the front and introducing the programming task as an implicit activity [12].

The second characteristic of our approach is the possibility of performing some scripting activities through acting and recording of abstract scenes. Instead of programming complex character animation episodes, users would be able to act and record these episodes and then manipulate them as animation libraries.

Acting and recording instead of typing program code is also related to the third characteristic of our approach, embodied interaction. Users will be able to use the physical world, including their own body, as the interaction medium. Recently, commercial digital games employing a similar interaction paradigm have appeared on the market and are already enjoying a significant amount of success [4, 16, 6].

2 Embodied interaction in programming

There are at least four ways in which embodiment, or more generally any interaction modality, can be used in programming:

- By supporting auxiliary activities. An auxiliary activity is a non-programming activity necessary for the execution of the program. Capturing data or creating a graphical element that the program will use in its execution could be considered as examples of auxiliary activities. The auxiliary activity can be performed in a different interaction modality to the rest of the programming activity, for example, a textual programming language might use graphic elements created in a graphic modality or data inputed as an audio recording. Technological innovations have allowed some programming activities to become auxiliary, for example GUI front ends can now be produced in a graphical environment (by drawing them) instead of programming them.

- As the interface medium. The interface medium is the way by which the programmer interacts with the program. Usually the interface medium is keyboard and mouse but programmers could also interact with the program via speech recognition for example.

- As the programming environment. The programming environment provides a set of tools to support the programming activity. Some examples of programming environment tools are coding editors, output windows, visualisations of the program, automated testing facilities, etc. Programming environments nowadays are usually graphical, even for textual programming languages. Logo and its tangible turtle [10] are an example of a textual programming language combined with a programming environment employing tangible elements.

- As the programming language itself. In this case the lexical elements of the language are expressed in the referred modality. Traditionally programming languages are textual but there are also visual and even tangible programming languages. One could think of a tangible programming language within a graphical programming environment, or of a textual language within a tangible environment, although this might not make much sense in practice.

The research programme we are undertaking is related to investigating the ways in which embodied interaction can be used to explore computing concepts and, for each way, the advantages and disadvantages it might offer over traditional methods. So far we have identified one such way, the acting and recording of abstract scenes. This can be considered as an auxiliary activity which can replace a programming task (the coding of animation scenes). We believe there are several benefits of employing embodied interaction in this way for educational applications: it can make sophisticated authoring tasks more accessible to a wider audience, it can establish a link between computing and other parts of the curriculum such as drama,
English and the performing arts, it could help to improve interest in computing degrees and careers and could also enable a more active interaction in the physical sense.

3 Acting and recording abstract scenes

The main idea behind the approach is the possibility of recording abstract scenes or movements. This is done through a combination of sensor tags placed on different parts of a person’s outfit, several video cameras and a large screen. This arrangement works as a magic mirror, users will be able to see a reflection, only that this reflection can be in the form of a character of their choosing (see Figure 1). Additionally, users will be able to record their movements as if they were recording a film with a video camera. However there is an important difference: when users employ a video camera they record scenes, when they employ the magic mirror they record movements. Movements are more abstract than scenes in that there are several parameters that have to be specified and that could be modified. Some of these parameters are the character doing the movement, the background or location where the movement is taking place, the size of the character performing the movement (relative to the background), etc. These recorded movements can be considered as units of processing similar to programming subroutines. Instances of movements can be duplicated, speeded up, played backwards, connected to create composite movements, etc. Concrete scenes can be populated with several instances of movements and each one of these instances can be associated with specific behaviours when players interact with them.

The editing of scenes therefore has to consider at least two aspects, movements as part of the space in a scene and movements as events in time. One option to structure this aspect of the environment is to follow conventions similar to those of commercial authoring environments such as Macromedia Director [15] or Flash [3] but providing an embodied form of interaction similar to platforms such as the Eye-toy [4] or the Nintendo Wii [16]. Another option is to employ tangibles, working on a table top for example. Ideally, the interaction paradigm for the manipulation of movements should be similar to their recording (an embodied style of interaction or at least compatible with it).

The proposed approach will therefore be implemented as a platform (the Stage Platform) comprising the magic mirror to record movements and an additional environment to manipulate them. Because of their high level of abstraction, movements are a natural way to support game authoring. We therefore expect that the majority of applications developed for the platform will be game authoring environments, although conventional games and even authoring environments for creating free-form playful application could also be developed for it. In this sense, the platform could be compared to a theater stage, plays can be performed on it in a similar way that game authoring applications can be executed in the platform. Actors perform a play in a similar way that meta-players build a game. Plays are usually performed for an audience, similarly, the product of a game authoring application is a game that is aimed at players. The Stage Platform would be similar to other generic platforms and tools like the DART.
Toolkit [8], the standard toolkit for programming augmented reality applications, in that it will be open (and we would actively encourage) for other research groups to use it. We feel that there is a need for a platform to enable developers to build applications with an embodied style of interaction and the Stage Platform can respond to that need similarly to the way the DART Toolkit provided developers with a tool to build Augmented Reality applications.

The game authoring environments developed for the Stage Platform, as well as the games produced by them, could have an educational component or have mostly an entertainment aim. One interesting question is related to whether the act of authoring games with this approach will have an intrinsic educational value for the meta-players regardless of the type of games produced.

3.1 Two scenarios

Young people will therefore use the game authoring environments to build their own games. These game authoring environments will be fairly specific so that from the outset young people focus on the environment’s application area, making the scripting and authoring inherent aspects of the task. This approach has also the added value that a good deal of basic functionality and libraries specific to the environment’s application area can be built-in in the authoring environment, providing enough support for young people to create more powerful games but at the same time leaving them enough room for a sense of challenge and the chance to be creative. Two example authoring environments that could be developed for the platform can be illustrated by the following scenarios.

Scenario 1. Dance Along. Caitlin, Camila and Jon from class 10C have watched Romeo and Juliet for their GCSE work and have got very interested in the courtly dancing. They use the Dance Along game authoring environment to recreate one of these dances and then dance along to it. First, they act and record basic sequence of moves of the dance using the magic mirror. Then they edit these dancing scenes (assigning characters, connecting and duplicating them) to create a complex dance with 80 characters. Finally they invoke a built-in functionality for producing a score for players (this works by comparing the trajectories of players dancing along with those of the programmed characters). Then they invite Chloe, who is also in 10C, to see the movie of the dance and dance along to it. Chloe chooses the character she wants to be and then dances in front of the magic mirror as if she were that character. When the dance finishes she looks at her score and watches the movie of her dancing as the character and realises that she didn’t do that badly in the dance.

Scenario 2. The Interview. Jade (12), Anne (13) and Johan (13) from 9LAN like playing computer games but enjoy even more creating their own computer games and having their friends play them. As they’ve been looking at Shakespeare’s “Much Ado About Nothing” at school, they use the authoring environment called The Interview to create an application that will allow them and their friends to practice ‘hot-seating’ as a way of developing Beatrice’s character. First they act and record basic scenes with Anne playing the role of the interviewer, a masked cartoon character. Some of these basic scenes are the questions to be asked, gestures to be performed and comments that can be made at some points of the interview. Then they edit these movements to create a generic plan for the interview. This plan involves the interviewer character reacting to interaction with players as it will be manipulated similarly to the way a string puppet is handled (one player will interact with the character to ask the next question, make a comment or a gesture). When the plan for the interview is ready they decide that Johan will handle the interviewer and invite Helen, who is in their same form, to do the ‘hot-seating’. Helen sits in front of the magic mirror, appearing as a cartoon version of Beatrice and answers the questions posed by the interviewer character. The interview is recorded and it goes so well that they show it to the rest of the class.

The two scenarios exemplify the sort of authoring environments that can be developed for the platform, the types of applications that young people can create with the authoring environments and the way these applications can be played. Note that the first scenario illustrates a game-authoring environment while the second exemplifies an environment used to build a more free-form playful application (there is no scoring mechanism for example). This illustrates the versatility of the platform. This versatility could be helpful in evaluating the motivational potential of the approach against a variety of
environments. Also, in both cases both the meta-application and the application are played with an embodied style of interaction. This is the ideal situation but it could also be the case that the game is played as a conventional desktop application.

It is easy to see the educational potential offered by game authoring environments developed for the platform. The second scenario, for example, could be used to interview not only characters in plays but also history personalities or even geometric shapes in maths or animal cells in biology for example. Also, in the two scenarios both the meta-game and game support curricular topics. However this does not have to be the case, it is up to the game authoring environment to constraint the activities meta-players perform and the types of games they produce.

### 3.2 A proof-of-concept prototype

We have developed a prototype of the magic mirror with limited functionality as a proof of concept. This prototype only implements the reflective part (people can see a ‘reflection’ of themselves as a cartoon character) of the approach at the moment. It cannot be used to record scenes yet and characters are presented in a two-dimensional environment. It has been built using the AR Toolkit Plus [18] using C++. The recording of scenes do not seem to comprise any significant challenges but presenting fluid and anatomically plausible movements might require the use of motion capture techniques. At the moment we are conducting workshops with school teachers to obtain feedback on the potential of the tool and will also be conducting a limited amount of user evaluations.

### 4 Conclusion

This paper sketches a research programme related to investigating the ways in which embodied interaction can be used to teach and learn computing concepts and for each way what advantages and disadvantages might it offer over traditional methods. The paper presents a tentative analysis of the ways in which embodied interaction can be used in authoring and programming, explores the potential of a specific approach, the acting and recording of abstract scenes, and briefly describes a prototype built as a proof of concept.

Future work of this research will consider a full implementation of the platform as well as of educational applications similar to the ones described in the scenarios of Section 3.1. Additionally, the research will explore some of the other ways (described in Section 2) in which embodied interaction can be used in programming.

### References


