Game-Based Concept Visualization for Learning Programming

Frederick W.B. Li  
School of Engineering & Computing Sciences  
University of Durham  
United Kingdom  
frederick.li@durham.ac.uk

Christopher Watson  
School of Engineering & Computing Sciences  
University of Durham  
United Kingdom  
christopher.watson@durham.ac.uk

ABSTRACT
This paper presents a game-based learning environment to support novice students learning programming. The proposed environment exploits game construction tasks to make the elementary programming more intuitive to learn. It also comprises concept visualization techniques, which allows students to visualize and learn key concepts in programming through game object manipulation.

Categories and Subject Descriptors
K.3.2 [Computers and Education]: Computer and Information Science Education—Computer science education

General Terms
Design

Keywords
Game-Based Learning, Programming Education

1. INTRODUCTION
Learning to program is typically a fundamental skill required to obtain a Computer Science degree. However, many students find the process of learning programming language is difficult and unpleasant. Programming courses generally are accompanied with a notorious reputation for their difficulty and failure rates, as highlighted by a recent multinational survey of 63 institutions which found on average a 67% pass rate in CS1 courses [5]. Other studies have highlighted that even students who have completed the course may still not know how to program [19] or how to interpret and complete small code fragments [15]. Even by the point of graduation, students may still struggle and fail to design software systems [8][16].

Several theories exist as to why programming education can be difficult. Some can be attributed to the nature of programming itself. Programming is not a single skill but a complex cognitive activity, where a student must simultaneously build and apply several higher order cognitive skills (such as abstraction), in order to solve a particular task [26][22]. Other reasons are associated with the traits of individual students, such as lack of motivation [9], becoming impatient with the lack of “immediate results” from applying little syntax [10]. Others may be unable to create a mental model of how their program relates to the underlying system[4] or a clear model of program flow [20]. Further reasons are associated with the choice of language and teaching methodology used.

In this paper, we have presented a game-based learning environment to support novice students learning programming. Our main contribution is blending programming learning tasks with the game construction process using tile-based games. Surpassing existing authoring-based game-based learning methods, we provide methodologies to facilitate the learning of use of variables/data structures and the construction of simple programming logic. The remainder of this paper is organised as follows. Section 2 outlines related work and the uses of games within programming education. Section 3 presents our proposed game construction method. Section 4 outlines implementation detail of a prototype. Section 5 presents a preliminary evaluation, and finally Section 6 concludes the paper.

2. RELATED WORK

The usage of computer games in introductory programming courses can be divided into three main categories. The systems and tools vary in the nature and scope of the metaphor they use, the language they teach, the quality of their appearance, and the range of concepts designed to convey. An overview of systems is presented in Figure 1.

2.1 Authoring-Based Approach

This approach uses computer game development as the student’s main learning activity. This constructivist technique has shown notable success in previous research as students generally demonstrate a greater understanding of programming concepts, self belief in their abilities, satisfaction and higher retention of material, in comparison to learning through traditional lectures and exercises [5][13].

For instance, in [21], students are provided with game-themed assignments along with a set of development libraries and possibly a partial implementation. The students then learn programming concepts by either constructing, completing or modifying an existing game, where the developed game becomes a reward for the student’s learning progress.
This method is in effect sugar coating the learning process for motivating students. However, students will still encounter traditional difficulties (such as IDE complexity or cryptic compiler messages) during the early stages of development [24]. Additionally if constructing a game from scratch, the students would be unable to visualise the effects of code execution until the basic structure of the game is implemented. Therefore, it does not always support novices in developing the mental models crucial to success [25].

Alternatively, students are provided with a highly graphical and simplified learning tool where they can gain an understanding of programming concepts through direct media creation and manipulation. Usually systems of this nature attempt to decrease the students external cognitive load [14] by allowing program construction through the use of block-based graphical languages, therefore allowing a student to focus upon problem solving aspects rather than syntax errors. For example, Alice 2 uses animation, storytelling and game construction to introduce students to object-based concepts. Students specify programs visually through a series of drag and drop functions and then view the program execute in a 3D micro-world [7]. A similar system is Scratch which allows students to create simple games by scripting the actions of sprites using drag and drop code blocks [17]. Whilst these examples have shown some success in teaching basic programming concepts, a student is still required to eventually scale-up to a more complex programming language. Greenfoot tool attempts to address this issue by providing a framework allowing novices to author and interact with their own 2D micro-worlds by using Java. The system aims to visualise object orientated fundamentals (such as object creation from a class) clearly to novices by extending the idea of a BlueJ object bench further into a object world. Novices can specify their own World and Actor entities whose methods can be invoked, and state inspected, in a similar manner to BlueJ. These entities are then added to the micro-world and their logical behaviour defined by completing the act() method. Generally the learning tasks of the system is for a user to either modify or complete a partially implemented scenario [12].

2.2 Play-Based Approach

Another approach is to convey programming through game playing. A game usually consists of a series of missions which are related to specific concepts. A student then gains knowledge of these concepts by developing programming strategies to complete these tasks and to watch the strategy executing within the environment. Although these systems may require a student to develop code as part of their solution, they differ from the authoring approach by placing a smaller emphasis on coding. The code required is usually restricted to completing a function or to specify commands to control a game entity. The majority of solutions in this category are also targeted at specific concepts, rather than a full language like most of the authoring approaches.

For example, in PlayLOGO 3D children had to destroy their opponent by entering a sequence of syntactically correct LOGO-like commands, teaching them the concepts of method calling and parameter passing [23]. Other solutions required students to complete functions or debug existing code to overcome game obstacles and tasks. In The Catacombs the player assumed the role of an apprentice wizard, and had to use his spells (programs) to overcome obstacles such as locked doors [2]. Saving Sera required a player to rescue a kidnapped princess, by debugging and constructing a series of code snippets, learning about recursion and control structures in the process [2]. Elemental: The Recurrence also taught students recursion by providing visualisation of
a depth first search and stack operations within a 3D micro world [6]. Prog & Play and Resource Craft adopted a different approach by providing generalised scenarios by using a multi-player real time strategy game rather than providing a single set of missions to complete. This had the advantage of providing potentially unlimited game scenarios [11][18].

Other solutions removed the need to type syntax altogether and instead allowed students to solve game tasks by using pictorial representations, thereby placing an emphasis on the underlying concepts and problem solving skills instead of syntax specifics. LightBot\(^1\) is a puzzle game where players must navigate a robot around a grid-based environment illuminating certain tiles. Programs are constructed from a set of pre-defined command tiles however each function, including the main function, can only hold a limited set of commands. This eventually forces the player to use additional functions for repetitive actions or to develop recursive solutions. A similar system is Robozzle\(^2\) which in addition to function calling can teach players the use of if statements by using different coloured tiles.

2.3 **Visualisation-Based Approach**

The third approach lies between the authoring-based and play-based approaches. The systems in this category do not convey programming concepts through game construction. However they also cannot be classified as games as they lack game elements such as scoring, competition or story. Rather these systems use micro-worlds for the purpose of concept visualisation and to demonstrate code execution in a visual context. Turtle-based approaches are an example of this type of system. C-Sheep [1], Jeroo [27] and Karel the Robot [31] can demonstrate the effect of code execution to students. But they cannot be classified as game-play either as they lacking game elements. The objective of the player is to simply observe a turtle moving around an environment. There is no story, goal, or learning task to solve. Our previous work [30] falls into this category.

3. **OUR METHOD**

Generally, the process of learning programming concepts involves four stages:

1. **Receiving:** This could be passively received from a lecture or from reading course materials, e.g. arrays store a set of values.
2. **Visualising:** This uses visual aids to help student understand and construct the mental model of a concept, e.g. showing how values are indexed in the array by using a diagram.
3. **Reinforcing:** Students are allowed to practice with the concepts, e.g. creating a new array.
4. **Applying and Synthesising:** Students are involved in constructing applications using what they have learnt.

Our work aims to support the learning experience of programmers at each of these four stages, by presenting concepts in a manner which makes them more intuitive and easier to understand. Our method is to blend a simple game construction process with a set of inter-connected programming tasks, such that the student can gain knowledge of the related programming concepts through task completion.

Our method offers a more **intuitive** and **comprehensive** approach to learning within programming education:

1. **Intuitiveness:** Game construction tasks allow students to graphically experiment with numerous programming concepts. The meaning between syntax and output can be clearly viewed, increasing the likelihood that it will be understood.
2. **Comprehensiveness:** We allow students to learn programming concepts comprehensively through the game construction process. Unlike existing authoring-based methods, such as Greenfoot [12], which only conveys programming as mainly comprising object creation and manipulation, we also provide methodologies to facilitate the learning of use of variables/data structures and the construction of simple programming logic.

This differentiates our method from most existing game-based learning methods. We believe that by closely aligning game and program construction tasks, and combing the process with meaningful graphical visualisation, can provide a deeper impact towards effective learning.

In a sense the intention of our approach is similar to that of Greenfoot [12], by emphasising the learning of programming concepts through the use of a “real” programming language (Java). On the other hand our approach differs in the problem solving aspect. Greenfoot mainly aims at conveying the basic concepts of object creation and manipulation through a series of drag and drop functions. In terms of game construction, aspects such as collision detection are already managed for the student, who can specify actions to take on events through the use of an API. The environment would be best described as an exploratory learning tool, where students can create and interact with visual objects and inspect their state.

Our approach however is more closely orientated towards problem based learning. It aims to convey programming concepts to students through a series of carefully designed game construction themed tasks, such that the students can gain an understanding of programming concepts through task completion. Unlike Greenfoot, we have chosen to use a simple tile-based game\(^3\) as a platform to allow students to experiment with programming concepts. We have chosen this as it is simple in nature (2D array) and is therefore easy for novices to understand. It also provides a regular structure, allowing students to visualise results in a well organised way. Furthermore, whilst a tile-based game is simple in structure, it is also flexible enough to allow different types of games to be developed based upon the same underlying principles. For example, assigning different values to cells in the array, changing the graphical representation of the values or modifying the collision logic can yield an entirely different scenario. Therefore the use of this structure allows enough room for the student to apply their newly learned knowledge and to build creative applications - both important for motivation and to support more advanced learning.

Generally, many game construction tasks are well suited towards programming education, with multiple construction methods.

---

\(^1\)armorgames.com/play/2205/light-bot
\(^2\)robozzle.com
\(^3\)www.tonypa.pri.ee/tbw/
tasks forming a natural parallel with key programming concepts (Table 1). Our approach is to use construction themed tasks as a means to convey programming concepts. For instance, if the student was to create a car racing game then one of the construction problems could be to determine how to create the racing track. This can be easily solved using the tile-based approach by defining a range of values to represent different surface types (e.g. 11 = grass, 22 = track) and then assigning them to various cells in the array. This allows the concept of variable assignment to be conveyed to students in a simplified but meaningful manner. Another construction problem could be to work out how to move a car across the screen. If the student was taking a more advanced graphics course, then they would have to program complex collision detection routines based on mathematical functions. On the other hand, with the use of our tile-based game structure, the solution of performing collision detection can be simplified to checking the content of adjacent cells (inspection of data structure) and to determine whether or not to move the car (flow control).

4. IMPLEMENTATION

We have implemented a small Java-based prototype system based on our method. In order to start learning each student needs to create a project and is provided with two initial classes. A Board class is used to represent the tile-based board game in the format of a 2D array. The second class is the Game class which is used to manage the overall functionality and settings within the game. For example, tasks such as screen update are defined within this class as part of the main game loop. We discuss three examples of problems, how they can be solved, and the concepts which they convey. We continue to use the theme of a car racing game in our examples.

4.1 Object Creation

Sample task: The student is required to create their own racing track, and to add a car to it.

Method and concepts: In order to create their own game, the student needs to add their own objects to the environment. This can be performed by implementing the Entity interface and adding their newly created Car class to the 'Objects' folder. To specify the visual appearance of the car, the student needs to implement two versions of the draw method. The first version accepts no parameters, and is used when the student wishes to create a static piece of background or scenery (e.g. the track). The second version is used for event handling and specifying changes in appearance based on keys pressed. This version consists of a series of if-statements where each event is mapped to a specific image name. More advanced students can create their own animated Sprites directly by using an Animation object and specifying a series of images and time intervals (as an array of strings). Currently the only other information which must be stored in the object is its position on the game board, stored as two separate integer values (x, y). The position can be defined by either specifying this information directly as cell indexes (Figure 2) or to specify the objects location by clicking on the location on the game board where it should be created (Figure 3). Once the appearance has been defined the student can then add the objects to the environment by adding it to array of Entities in the game class (either directly or via declaring a field).

To solve the problem the student first implements a new Surface class by extending the Entity interface and completes the draw() method to specify its appearance. They also decide to extend the class functionality by adding fields to specify the level of friction and surface type and update the constructor accordingly. The student can then create the track by adding Surface objects to the environment by clicking on the relevant location on the grid. This also assigns the Surface objects to the cells in the game board, and updates their position with the corresponding (x, y) cells which the user has selected. A similar method is used to define the position and appearance of the car, except the student will implement the draw() method which accepts KeyEvent as a parameter in order to ensure the car faces the correct direction. Even at this early stage, a number of concepts have been conveyed to the students. The idea of creating objects from classes, modifying state through method calling and variable assignment.

4.2 Game Logic

Sample task: Program the game so that a race only lasts for a limited number of laps and disqualify cars which leave the track.

Method and concepts: Generally specifying game logic can be used to convey the concepts of simple programming logic and flow control. In particular the use of looping constructs (either conditional or counter based) to perform repetitive actions can easily be conveyed through the use of the main game loop. To complete this task the student is required to write a function to test whether or not a car has competed a lap (programming logic) and then to decrement a lap counter accordingly (use of variable). They decide to terminate the race when the three laps have been com-
completed (conditional loop). They also decide to specify rules to affect the car if it crosses a wet surface or grass (programming logic) by making it come off the track (Figure 4). The student can verify the correctness of their logic simply by observing whether or not the intended outcomes occur based on conditions in the environment. If the car does not perform as expected, or completes too many laps then the students can understand that there is a problem in their code and refine it accordingly.

4.3 Game States
Sample task: The student is required to keep track of the number of races they have won.

Method and concepts: In order to solve the problem the student simply has to add variables within the Game class in order to track the number of times they win a race, and to increment the counter each time they win. Within the game method, they can then also specify drawing instructions to log the state information onto the screen. (Figure 5).

5. INITIAL EVALUATION

To assess the suitability of our approach, we have conducted a user study on a prototype. The main research questions we asked included: Can the approach convey concepts in a clear manner? Would the technique presented provide better support to students in understanding programming concepts than through traditional lectures and static materials?

The participants included six members of the universities Learning Technologies Team (LTT). The LTT regularly provides training and advice to staff on aspects of e-learning and is involved in developing tools to support education, such as Blackboard extensions. Participants were invited to either a group or individual session which lasted for about 30 minutes. Each session included informal discussion aided by prototype screens.

Quantitative feedbacks collected were mainly positive. The game-based concept visualisation of coding concepts such as method calling proved to be a popular idea to support
novices and the mapping between source code to actual game events was viewed as an effective method of demonstrating the effect of code execution to novices. Overall, participants indicated that they saw value in the techniques for teaching novice programmers with the majority indicating the approach would be appropriate for novices in picking up an understanding of elementary concepts.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a method to teach introductory programming concepts by blending the programming learning tasks with the game construction process using tile-based games. We believe that this will make a deep impact to programming education. At the moment, we have not yet conducted a proper student evaluation as students had left for the summer holiday. We will keep this as a future work while we have been improving our prototype by enhancing its features and interface.

Acknowledgement

This project was partially supported by a research project on multi-resolution modelling [28][29].

7. REFERENCES


