UNIVERSITY OF CAPE TOWN

DEPARTMENT OF COMPUTER SCIENCE

COMPUTER SCIENCE HONOURS
FINAL PAPER
2015

Title: PyTeacher: Designing a Visual Programming Environment and Evaluating its Flow Experience

Author: Mark Grivainis

Project Abbreviation: PyRob

Supervisor: Gary Stewart

Category	**Min**	**Max**	**Chosen**
Requirement Analysis and Design	0	20	10
Theoretical Analysis	0	25	0
Experiment Design and Execution	0	20	10
System Development and Implementation	0	15	10
Results, Findings and Conclusion	10	20	15
Aim Formulation and Background Work	10	15	15
Quality of Paper Writing and Presentation	10		10
Adherence to Project Proposal and Quality of Deliverables	10		10
Overall General Project Evaluation (*this section allowed only with motivation letter from supervisor*)	0	10	0
Total marks		**80**	**80**

PyTeacher : Designing a Visual Programming Environment and Evaluating its Flow Experience *

Mark Grivainis
Reseacher
University of Cape Town
Rondebosch
Cape Town, South Africa
markgrivainis@gmail.com

Gary Stewart
Supervisor
University of Cape Town
Rondebosch
Cape Town, South Africa
gstewart@cs.uct.ac.za

James Gain
Co-Supervisor
University of Cape Town
Rondebosch
Cape Town, South Africa
jgain@cs.uct.ac.za

ABSTRACT

Learning to program is a complicated process, as such PyTeacher was developed to provide a simple and enjoyable system to teach basic programming concepts. In order to accomplish this, an educational game was developed that contained a code-based Visual Programming Environment (VPE). A series of questions was then devised to determine if it accomplished its goals. Survey results indicate that one of the genres most entertaining to students, puzzle solving, was also well suited to the educational goals required by the game. The puzzles in PyTeacher are presented in the form of mazes which the user is required to navigate successfully. Successful navigation requires the user to write Python code which utilizes commands that enable the puzzle to be completed. To successfully navigate a maze these commands are performed in a certain order, which requires the use of algorithmic thinking. Results of user testing show that students embrace the idea of a game to aid them in their first year coursework and find the experience both enjoyable and educational. These results indicate that the game achieves its purpose in promoting an exciting and engaging educational experience for the students.

1. INTRODUCTION

1.1 Problem Statement

Learning to program is a daunting task for any student. Fundamental concepts such as loops can be hard to visualise, as programs return the end result without showing the means by which this is obtained. Visual Programming Environments (VPEs) offer a means to teach these concepts in a way that permits students to follow the steps of their program while it executes. This is achieved by allowing students to see which line of code is currently being executed and offering a visual means for observing the state of the program. The use of VPE as a means to teach introductory programming to students with no experience has resulted in these students achieving better grades than if they were taught using traditional methods. Students taught using VPEs are also more likely to continue with second year Computer Science [10]. To go one step further, using VPEs in a game format would enable students to learn programming in a fun environment. The games would need to be entertaining and challenging to the students; challenging enough to keep them interested but not too difficult to cause a state of anxiety or frustration. Ultimately when developing a game, one desires the user to be totally involved in this endeavour resulting in what is termed a State of Flow. The higher the students' state of flow, the more engaged they become in the activity, thereby increasing their concentration and learning ability.

1.2 Research Question

Can an educational game using a code based learning environment lead to a state of flow?

1.3 System Overview

The system that was developed, PyTeacher, was split into two components. The front-end which consists of a user interface, allows students to interact with the game. Students can input code into a text editor and execute the code to see the result in the Game Window. The back-end parses the code that the student has entered and instructs the avatar to perform the relevant actions. An avatar is the representation of the player in the game world, in the case of PyTeacher, a quadcopter.

2. BACKGROUND

In order to investigate the research question regarding flow, insight into why Python is a good language for new students as well as an understanding of the different types of VPEs that currently exist was required. In addition, a way of measuring and defining flow experience needed to be researched.

2.1 Python for students new to programming

When first learning to program, emphasis should be placed on thinking algorithmically not on the intricacies of the language in which the students will be programming. Java, C++ and C# all have a complicated language components to understand before students can write structured programs. In contrast Python is a high level scripting language that is far simpler to learn when compared to Java, C++ or C#, although it will achieve the same end result [5]. A 'Hello World' program is usually the first program that students encounter. It is simple and is often used as a test to ensure that the programming environment is correctly configured. Running this program results in the words 'Hello World' being displayed by the console.

*Appendicies are available at http://pubs.cs.uct.ac.za under Honours/2015/PyRob. The project web-site contains a page dedicated to the appendices

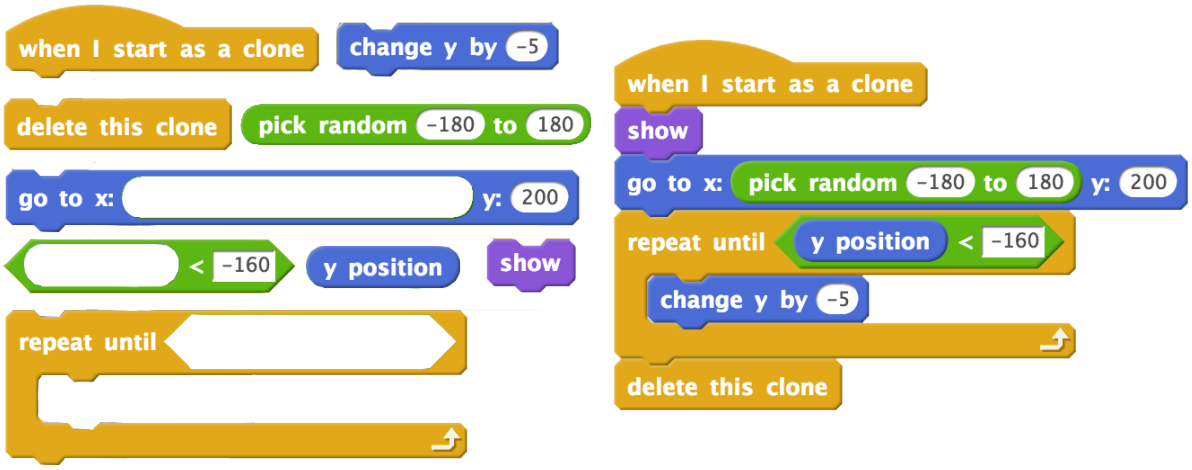

Figure 1: A demonstration illustrating how Scratch uses separate code blocks (left) and allows users to combine them to form a working function (right).

As an example we can compare the code to make a 'Hello World' program in Python versus Java:

Python

```
print ("Hello World")
```

Java

```
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello, World");
  }
}
```

It is clear that students who are instructed in Java will need to have a basic understanding of far more concepts, such as classes and access modifiers, to perform simple functions. While Python has the capabilities of utilizing classes, these are not required to write simple programs. Because of the ease of use, Python was chosen as the programming language.

2.2 Visual Programming Environments

Traditionally programming is taught to students using a Software Development Kit or SDK. The issue with this approach is that students who are learning basic algorithmic thinking will only use a fraction of the SDK. In contrast VPEs offer the functionality for students to learn these programming concepts in a more visual environment. There are two different types of VPEs, Drag and Drop or Code Based. Drag and Drop based VPEs use blocks to represent the different components required to build a program. These blocks are constructed in such a way that they can only fit together in ways that form a working program (**Figure 1**). In contrast Code Based environments require students to input code which will result in the system performing the required action. This can be in a certain language, or allow the user to choose from a range of languages. An understanding of the types of VPEs that already exist and have proved successful in educating students with regards to learning algorithmic thinking was required, in order to incorporate these aspects in this system.

2.2.1 *Drag and Drop Interactions*

There are a number of VPEs that utilize Drag and Drop interfaces. Scratch and Alice described below are two of the most successful.

Scratch is an after-school educational game that is aimed at a younger audience ranging in age from 8 - 16. The user interface consists of a command palette, scripting window, sprite selection window and stage. Blocks are selected from the command palette and linked together in the scripting window which results in sprites performing actions on the stage. Even though Scratch is aimed at a younger audience, there is a large community of adults who also contribute work to the online repository [6, 9].

Alice was developed as an application that would work in conjunction with Computer Science courses. It is aimed at an older audience ranging in age from 12 - 19. Alice is based on Java and provides a platform for students to progress into using Java as a programming language. The commands are also represented as blocks, which take various parameters to perform functions. A key difference between Alice and Scratch is that Alice features a 3D stage in which actors perform tasks. A study conducted at Saint Joseph's University demonstrated that the use of Alice in teaching first year Computer Science students, who had no previous experience in programming, resulted in these students achieving similar grades to students who had previous experience in programming, and a large number of these students decided to continue with second year Computer Science [3].

2.2.2 *Code Based Interactions*

Instead of using blocks to represent coding concepts, Code Based Interactions require users to enter actual code to interact with the game. The advantage of this is that it takes users less time to adapt to an actual programming environment but comes at the cost of having a slightly higher learning curve.

Logo was designed in 1967 and is mainly remembered today for its use of Turtle Graphics in which a user can draw lines by giving a turtle commands to move around the canvas. While it has been used to teach basic programming

logic, it is equally suited for teaching mathematical thinking [1]. While Logo is no longer commonly used as a programming language, Turtle Graphics now comes as a default package included when installing the Python programming environment and is still used for teaching introductory programming.

Greenfoot is a Java based VPE that is aimed at high school students with some programming experience. Greenfoot contains features such as an object inspector, pop-up menus for objects and object state monitoring [7]. It has a built-in editor compiler and debugger. The key advantages of Greenfoot as a platform are its flexibility, expandability and social interaction [8].

As students will be programming primarily in Python and using PyTeacher as an additional tool, Code Based Interactions are more appropriate as the students will not be required to learn two different methods in order to write programs.

2.3 Flow Experience

Flow is a mental state that is entered when one is fully absorbed in the activity that one is performing. Whether these activities offer an external reward or not does not affect whether a flow state is achieved [2]. It is often described as a feeling of being 'in the zone', where actions are performed almost of their own accord and there is a feeling of total confidence in the task at hand. The following components characterize a state of flow [4]:

1. A balance between perception of one's skills and the perception of difficulty of the activity (task demand). In this state of balance, one feels both optimally challenged and confident that everything is under control.
2. The activity has coherence, contains no contradictory demands and provides clear, unambiguous feedback.
3. The activity seems to be guided by an inner logic.
4. A high degree of concentration on the activity due to undivided attention to a limited stimulus field.
5. A change in one's experience of time.
6. The self and the activity are not separated, leading to a merging of the self and the activity and the loss of self-consciousness.

Flow experience is an important consideration to take into account when designing games as this will lead to players spending time playing the game and finding it enjoyable. In order to evaluate the flow experience that users experience while playing the game, the Flow Short Scale survey was used. This survey has been validated and shown to be successful in measuring the flow experience [4].

3. DESIGN

The main goal of this project was to design a VPE which would allow students to control an avatar through the use of Python code. It offers an interface that is easy to learn allowing students to begin programming immediately. The design can be separated into two parts, Experimental Design which describes the experiments that were conducted as well as the ethical issues that were posed, and System Design which gives an overview of how the system was developed.

3.1 Experimental Design

Qualitative research methods were used to assess students' perception of the game throughout development. The results of this research strongly influenced how the game was developed and helped in answering the research question. The user group consisted of first year Computer Science students at the University of Cape Town who were attending the CSC1010H course. This course is aimed at students with little to no programming experience which made them perfect candidates for evaluating the game. As user testing was performed towards the end of the academic year the students had have some experience in using the concepts that were required in order to play the game. As the primary goal of developing the game was educational, these students served as excellent candidates for evaluating the system and for provided meaningful feedback. Ethical clearance was required in order for the research to be conducted. Clearance was obtained from both the Science Faculty and Student Affairs. In addition students were required to sign consent forms when they agreed to participate in the testing (see **Appendix A** for the above-mentioned documents). A survey was conducted early in the development process to determine the number of students who play games in their spare time and the genre of game that they prefer. Later in the development process, user testing was conducted to evaluate the system and identify problem areas. A final set of user testing was then performed, the first part of the test was identical to the first set of user testing and in addition the Flow Short Scale survey was conducted. The same group of students was used throughout all of the testing.

3.2 System Design

A number of considerations needed to be taken into account during the development of the optimal system. The choice of which programming language to use was crucial as the students for whom the system was being designed only had experience using Python 3. In addition, this language allowed the use of its built-in parser which permits the code that students input to be converted into commands for the game.

In order to develop PyTeacher a Game Engine was required. Blender was selected for this purpose as it is an Open Source environment for 3D design, animation and rendering. In addition it is one of the only Game Engines that use Python 3 as its scripting language. While Blender offers the core functionality required to build a game, it does not provide any means of creating user interfaces. The use of BGui, which is an external library that was developed by Mitchell Stokes, provides this much needed functionality. It does this by providing a Python interface to QT libraries. This package offers a method to lay out user interfaces as well as providing elements such as buttons, labels and text input. An overview of the System Architecture can be seen in **Figure 2**.

The choice of design methodology can lead to the success or failure of a project. As we expected to be making a large number of changes to the system throughout the development life cycle, we adopted an Agile approach. This methodology follows an iterative process in which requirements are gathered and analysed. Using these requirements, goals are set, implemented in the system and tested before starting the process again. In order to track changes throughout the design process, a form of version control was

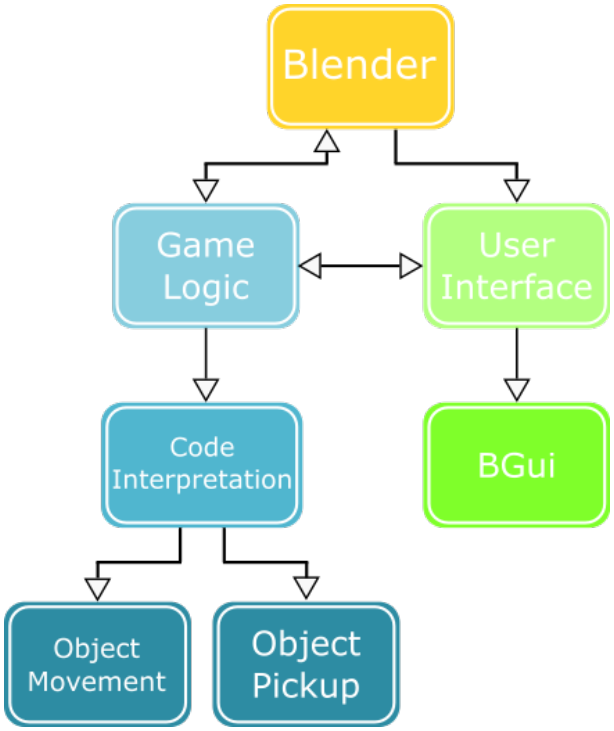

Figure 2: System Architecture Diagram. An illustration indicating the manner by which the separate components of the system interrelate.

required. Git was chosen for this purpose as it offers private repositories and readily allows changes to be incorporated during development.

4. IMPLEMENTATION

Feasibility testing was conducted before development of the system was started. The results showed that the participating students are interested in a game that would aid them with their coursework. Students were also required to fill in a survey regarding which game genres they enjoyed most. While racing games was the most popular genre selected by students, it is not suitable for teaching algorithmic thinking. The second most selected was the puzzle game genre which is far more suited to the task. A puzzle game that utilized mazes that students would be required to navigate in order to progress in the game was therefore developed.

Design documents were used to identify the key features of the game and any technical issues that might arise. Identifying these issues timeously results in risk mitigation as solutions can be formulated at an early stage. With these purposes in mind a Game Design Document was created. This gave an overview of the game and the tasks that users would be required to perform. In addition, a Technical Design Document which outlined technical issues posed by creating the game was prepared (see **Appendix B** for the above-mentioned documents).

Paper prototyping is a method of testing the system at an early stage using paper cutouts to represent all the elements in the game. Early designs of the user interface and gameplay mechanics were tested using this approach. This allowed for the testing of multiple layout configurations in

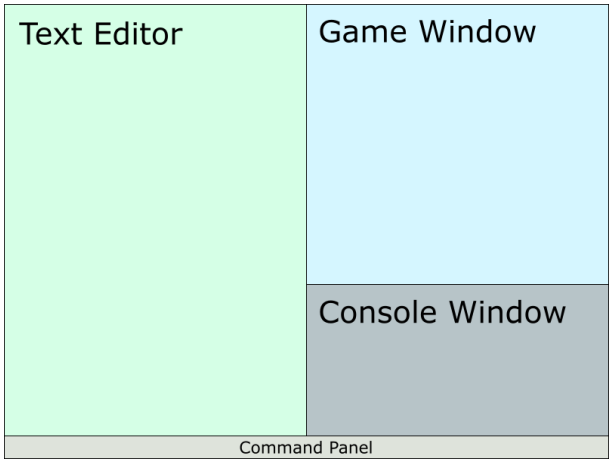

Figure 3: The interface design showing how the screen is divided into regions

order to find one which was intuitive. Keeping track of the state of the system throughout the process gave meaningful insight into how the final system would need to be developed. The Blender game engine serves as the foundation of the system. Game objects such as walls, the start location, goal location and avatar are laid out in scenes to create levels. A scene also consists of a virtual camera that is used to set up the view that the user will see and lights which are used to produce shadows and subtle lighting effects. Interactions with the game engine are achieved through the use of logic blocks. A logic block consists of a sensor which detects an event that takes place in the game, a controller which acts as a logic gate so that multiple sensors can be linked together and an actuator which performs an action in the game environment. External Python scripts act as controllers and actuators as they control the flow of the game as well as perform actions that occur in the game. While BGui allowed for the implementation of a user interface, the text input was only capable of single line input. The code for this element was modified to allow for multi-line input in order for programs to be written in the game. The user interface was split into four regions (**Figure 3**). Each region serves a different purpose: The Text Editor allows students to input code, the Game Window displays the maze to the students and shows resulting movements of the avatar during execution of the code, the Console Window alerts the students to any errors present in their code and the Command Panel offers buttons for executing the code and resetting the game as well as a progress bar which fills as steps are executed. During execution the Game Window was made full screen to allow students to follow how the avatar navigated the maze. User testing showed that this feature made it harder for students to follow the execution of their code as the Text Editor was hidden. Even though this made the system unintuitive students did enjoy seeing the results in full screen. With this in mind an extra button was added to the command panel that allowed students to change the way the game was displayed during execution. The default and zoomed views allowed for code to be executed with all of the user interface elements remaining in place. The default view did not change the way the game was displayed in the Game Window, while the zoomed view showed an enlarged

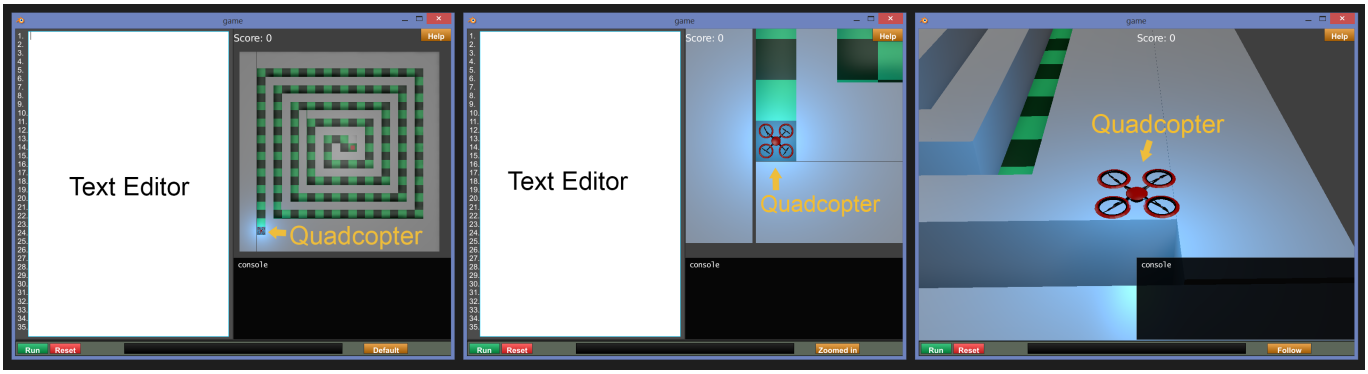

Figure 4: This illustrates the views available to students during the execution of their program. The default view (left), zoomed view (center) and follow view (right) are depicted.

version of the maze. A final option, called follow, allows the student to make the Game Window full screen during execution. The follow view gives a 3 dimensional view of the avatar while the other views provide a 2 dimensional top down view. These three views are depicted in **Figure 4**.

In addition, as testing indicated that students did not find the interface intuitive, separate colours were given to the buttons in order to differentiate their functionality. A series of pop-up windows was added to instruct students on how to use the environment. Another factor that led to the interface being unintuitive was that the Text Editor only allowed for the cursor to be moved using the keyboard. As the mouse is generally used for this purpose, students had difficulty editing their code. This issue was addressed during the next iteration of development. The final user interface is shown in **Figure 5**. The left half of the window was used as a text editor and had the solution to the maze shown in the right half of the window. Final user testing indicated that the changes which were made to the interface, and how students interacted with it, resulted in a very positive user experience.

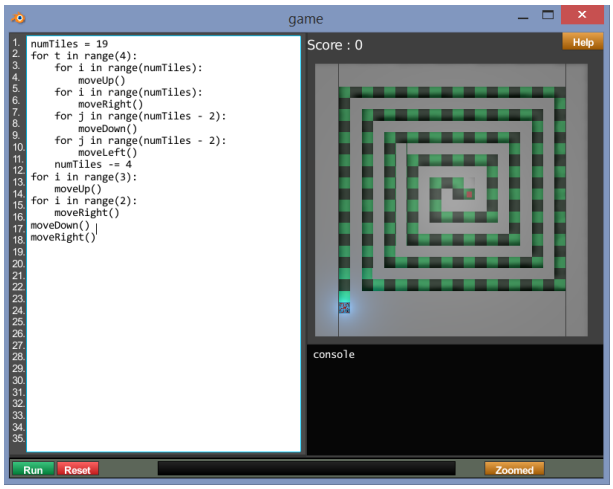

Figure 5: Final layout of the user interface.

Question	Yes	No
Do you play video games in your spare time?	12	2
Would you be interested in a video game that assists you in learning your coursework?	14	0
Is this year the first time you have been exposed to programming?	13	1

Table 1: Results of the feasibility survey

5. RESULTS

5.1 Feasibility Test

While developing PyTeacher, first year Computer Science students doing CSC1010H were invited to complete a survey and attend two user testing sessions. The survey determined whether students played computer games in their spare time, how interested they would be in a game that would assist them with the coursework required for first year Computer Science and what genre of games they enjoyed. The survey gave insight into how well the students would accept the game as part of their course, as well as helping us to determine a genre that would be inducive to learning in addition to being entertaining to students. Fourteen students participated in the survey the results of which are listed in **Table 1**.

Results showed that twelve students play video games in their spare time while only two students do not. This may indicate that if students enjoy PyTeacher they may play it in their spare time. As the course CSC1010H was designed for students who had had little or no previous exposure to Computer Science, the high number of students who had never programmed (13/14) was expected. All fourteen of the students indicated that they would be interested in a game that helped them with their coursework. The results from this survey therefore indicated that students would welcome a game that would aid them in learning first year Computer Science.

The results from the genre section of the survey are illustrated in **Figure 6**. Students were able to select multiple genres in which they were interested. The genre of game defines the core game play mechanics which are the actions that users are required to perform in order to suc-

Q#	Question	Most Common Response
1	Would you have enjoyed playing a game like this as additional material for your first year course?	Strongly Agree
2	Do you think that the visual feedback is a good indicator to what your code is doing?	Strongly Agree
3	Do you feel that there should be additional gamification elements?	Strongly Agree
4	Is this game something that interests you?	Strongly Agree
5	Is the layout of the user interface elements appropriate/intuitive?	Indifferent
6	Do you feel that the game challenges your problem solving skills in terms of designing a complete piece of code?	Agree/Strongly Agree

Table 2: Analysis of the Likert scale results for Initial User Testing

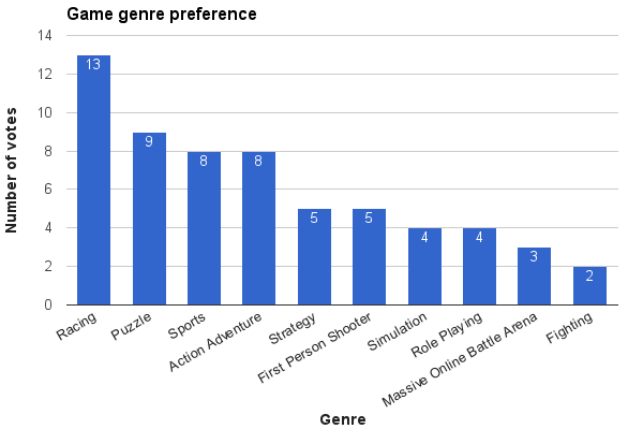

Figure 6: Depiction of game genres in which students are most interested

cessfully complete the game. As an example, in fighting games the core game play mechanic is to damage the other player without receiving damage yourself. The most popular genre chosen by students was racing games. The game play mechanics of a racing game are to drive faster than opponents and steer more accurately, avoiding obstacles that will slow you down. These mechanics are not conducive to an environment in which students will learn to program as they rely on the user having quick reflexes and making instantaneous decisions. Puzzle games was the second most popular genre. The game play mechanics can vary widely in puzzle games as the genre is broad. A puzzle game that incorporates chess might require players to checkmate an opponent given different board layouts in as few moves as possible. The key feature about these games is that they pose a problem which players are required to think about in order to come up with a solution. This factor makes them a good match for learning to program as the process that is followed when writing a program is similar. Educational puzzle games have been shown to increase a player's reasoning abilities as well as his/her strategic and tactical thinking [11]. Of the most popular categories, puzzle games were therefore most suitable for an educational game related to Computer Science.

5.2 Testing of Initial User Response

After developing a working prototype of PyTeacher, the system required testing and valuable user feedback. Students were required to navigate through all the levels of the game and then answer a short questionnaire. Seventeen students took part in this user testing session. Students could respond to questions using a Likert Scale by selecting a number between one and five. (One indicating that they strongly disagree, two indicating that they disagree, three indicating that they are indifferent, four indicating that they agree and five indicating that they strongly agree with the question). Likert Scale results are best evaluated using the mode which is the most frequent response to the question. Graphical representations of the data collected during user testing can be viewed in **Appendix C**. The results of the survey are shown in **Table 2**.

The results obtained after the user testing of PyTeacher were very positive and indicated that this may be a very helpful learning tool that could be beneficial to their learning experience. The response to Question 2 (**Table 2**) indicated that students strongly agreed that the system provided good visual feedback as to how their code was executed. This response was not expected as limitations posed by Blender prevented the line of code that was currently being executed from being highlighted. A possible reason for their 'strongly agree' response to Question 2 could be due to the students having developed an intuition into the outcome of their code.

As the system was a prototype it was lacking in game play mechanics. The objective was simply to move the avatar from a start location to a goal location through a maze. The layouts of the mazes were extremely simple at this stage which explains why students indicated a preference for additional gamification elements (**Table 2**, Question 3). Students showed an active interest in the project and helped identify problem areas (**Table 2**, Question 4). Many of the students remained after the testing was completed to ask questions about how PyTeacher was developed. A large number of issues were discovered with the user interface during testing (**Table 2**, Question 5). One of the problems was the lack of scaffolding to help guide students by instructing them on how to use the system. In addition, the Text Editor only allowed the cursor to be repositioned using the keyboard rather than the mouse, thus making it difficult to interact with the system. Additionally, students could not look at their code during its execution as the user interface was hidden while they ran their code. This was done in order to make executing the inputted code visually impressive as the entire screen would now show the avatar navigating the maze. While students did enjoy the full screen experience this hampered their ability to analyse their code during its execution. These factors all influenced the manner in which the students perceived the user interface. Proportionally high numbers of students agreed, and strongly agreed, that the system challenged their problem solving skills (**Table 2**,

Q#	Question	Most Common Response
1	Would you have enjoyed playing a game like this as additional material for your first year course?	Strongly Agree
2	Do you think that the visual feedback is a good indicator to what your code is doing?	Strongly Agree
3	Do you feel that there should be additional gamification elements?	Agree
4	Is this game something that interests you?	Strongly Agree
5	Is the layout of the user interface elements appropriate/intuitive?	Agree
6	Do you feel that the game challenges your problem solving skills in terms of designing a complete piece of code?	Agree/Strongly Agree

Table 3: Analysis of the Likert scale results for Final User Testing

Question 6). This shows that in order for the students to complete the game they needed to think about the problems programmatically and devise an appropriate solution, which is exactly what PyTeacher was designed to do.

5.3 Testing of Final User Response

After further development in which issues identified in the initial user testing were resolved, the same students were approached in order to test the final system. Of the original seventeen students, only nine assisted with the final testing. Students were required to navigate through the same mazes as previously in addition to a new maze which required far more complicated code to solve. They then answered the same survey that was used in the initial user testing (**Table 2**) as this would allow for a comparison between the responses to determine if there were improvements to the system. In addition, students answered the Flow Short Scale survey (see **Background** for description and below for results).

Once again students strongly agreed (8/9) that they would enjoy playing this game as part of their first year course (**Table 3**, Question 1). In the initial survey 9/17 strongly agreed with this question. However as the number of students that participated differed between the two surveys we cannot conclude that improvements to the system contributed to the increased fraction of students with a positive response to Question 1. As there were no additions to the system to aid visual feedback for code execution it is unsurprising that the same result to Question 2 was obtained (**Table 3** and **Table 2**). While students wanted additional game play elements (**Table 3**, Question 3), they no longer strongly felt that they were needed. This could be attributed to the addition of a new maze resulting in a far more challenging environment that gave students a better grasp of the type of challenges they would be expected to solve. Interest in the game is still high among the students with all students either agreeing or strongly agreeing that they are interested (**Table 3**, Question 4). In the initial user testing students felt indifferent about how appropriate and intuitive they found the user interface whereas the final testing showed that they now found it appropriate (**Table 2**, Question 5 and **Table 3**, Question 5). This indicates that the refinements made to the interface resulted in an improved user experience. Unfortunately BGui limited what could be accomplished when refining the user interface preventing us from making the program more intuitive. As the students overwhelmingly found that the game challenged their problem solving skills (**Table 3**, Question 6) the design of this game clearly succeeded in its objective. This indicates that the primary purpose of the software was achieved, namely

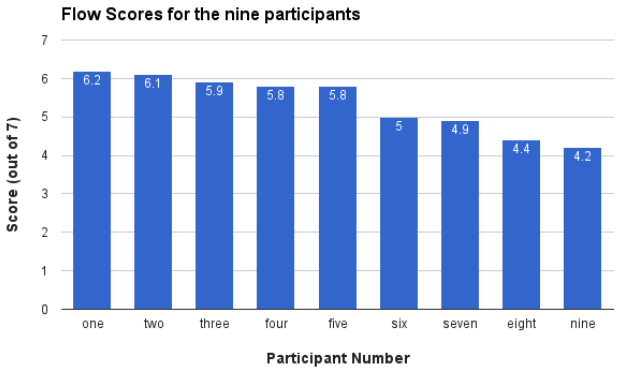

Figure 7: Flow short scale scores for the nine students

to develop an educational game that resulted in challenging the algorithmic thinking of the students while providing an enjoyable experience.

5.4 Flow Short Scale Survey

The Flow Short Scale Survey has been shown to accurately measure the level of engagement of the student with the task at hand, namely the student's flow experience. Students were required to complete the Flow Short Scale Survey during the final testing session. The survey consists of ten questions which are designed to measure a student's flow experience (**Figure 7**) and three questions that measure the perceived importance of the task (**Table 4**). Each of the ten questions targets one of the characteristics that leads to a State of Flow. Students record the level at which they experience each characteristic using a Likert-type scale ranging from one whereby they did not experience the characteristic, to seven, where it was strongly experienced. A flow score is calculated by summing up the results of all ten questions and dividing it by ten. The score then ranges between one and seven, where one indicates that there was no flow experience and seven that there was a strong flow experience.

Results show that flow is experienced by the nine participating students. Two students obtained a very high flow score that was slightly above six, four students obtained scores ranging between five and six and three students obtained a score between four and five (**Figure 7**). This high level of flow can be attributed to both the game playing experience and the act of programming. Programming is a form of problem solving and requires concentration and

		Number of responses per each level of the scale (1-7) for nine students						
Q#	**Question**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	Compared to all other activities which I partake in, this one is ... (easy (1) –> difficult (7))	0/9	1/9	0/9	3/9	2/9	1/9	2/9
2	I think that my competence in this area is ... (very low (1) –> very high (7))	0/9	0/9	0/9	2/9	2/9	5/9	0/9
3	For me personally, the current demands are ... (too low (1) –> too high (7))	0/9	1/9	1/9	5/9	1/9	0/9	1/9

Table 4: This table shows the results of three questions posed to determine the level of challenge (**scale 1 - 7**) faced by nine students while playing the game (**Q1**), their perceived skill in programming (**Q2**), as well as the demands which are posed (**Q3**).

understanding. This on its own can lead to a good flow experience. As solving the mazes was well within the students' capabilities while still posing a challenge their levels of anxiety would be low. The solution to the mazes required the use of programming concepts that the students had been taught in their lectures. This program effectively tested their skills in these concepts. A good flow experience requires having the correct balance of the required skills and an appropriately challenging task. These results indicate that the game achieved its purpose in promoting an exciting and engaging experience for the students. While the total flow scores were positive, four of the nine students indicated that they were not sufficiently engaged in the game (**Appendix C**, **Figure 7**, Question 1). This indicates that the mazes might not be complicated enough to challenge everyone. As the game only consists of a small number of introductory mazes, this is to be expected. The majority of the students (7/9) found the game challenged their algorithmic thinking (**Appendix C**, **Figure 7**, Question 10), indicating that the game achieved the purpose for which it was designed.

The final three questions of the Flow Short Scale survey are used to evaluate the degree of difficulty experienced by the students, how competent they feel in their programming abilities and the level of the demands posed by the game. The game was designed to challenge the programming ability of the students and the results indicate that this objective was achieved. A third of the students (3/9) rated the difficulty as medium (level 4) and more than half of the students (5/9) experienced varying levels of challenge (levels 5 - 7) while playing the game (**Table 4**, Question 1). All nine of the students felt that they were proficient programmers with more than half (5/9, level 6) feeling they were highly competent (**Table 4**, Question 2). While two of the nine students considered that the demands of the game were easy (levels 2 - 3) and one felt they were too high (level 7), the majority of students (5/9, level 4) indicated that they were perfectly balanced (**Table 4**, Question 3). These findings demonstrate that while the students are confidant in their programming skills, PyTeacher still challenges their abilities in the algorithmic thinking required to discover an elegant solution to the mazes, while not being overly demanding. The level of challenge posed by the game resulted in the students achieving a high State of Flow while playing PyTeacher and contributed to the overall success of the project.

6. CONCLUSION

The goal was to create an educational game using a Code Based Visual Programming Environment that would be challenging and exciting for students. Learning to program is a daunting task for any student as it requires time and practice to build an understanding of how programs execute. Visual Programming Environments assist students in achieving this understanding by visually demonstrating the actions that are performed during each step of the code they write. An early survey that was completed by students indicated that they were open to the idea of using a game to assist them in learning algorithmic thinking. In addition, this survey determined that many of the students enjoyed playing puzzle games which is a genre which has been shown to help students develop strategic and tactical thinking. This is the same kind of thinking that is required when approaching problems posed in Computer Science courses. The game that was designed made use of mazes that students would be required to navigate using core introductory Computer Science principles. Initial testing showed that students were required to think algorithmically and use these principles in order to find the solutions to these mazes, which demonstrated that the system achieved its primary purpose. Key issues with the user interface were identified during this initial testing session that assisted in developing a final interface which students found more intuitive and easier to use. The Flow Short Scale Survey was used to determine the students' level of engagement indicating their flow experience while they played the game. The results of this survey demonstrate that students had an exciting and engaging experience while playing the game. This indicates that the game is challenging and tests the students' skill at algorithmic thinking. These findings show that a State of Flow can be reached when using a game based Visual Programming Environment, demonstrating that students are focused and confident in their actions while playing the game. These results indicate unequivocally that an educational game using a Code Based Learning Environment can lead to a State of Flow.

During the development process it became apparent that while the Blender Game Engine functioned adequately, the need for the BGui libraries limited what could be achieved when designing the user interface. In moving forward it would be highly beneficial to redesign the game using another Game Engine such as Unity which offers a wider range of features for constructing user interfaces as this could further enhance the user experience.

7. REFERENCES

[1] A. Agalianos, R. Noss, and G. Whitty. Logo in mainstream schools: the struggle over the soul of an educational innovation. *British Journal of Sociology of*

Education, 22(4):479–500, 2001.

[2] M. Csikszentmihalyi and M. Csikzentmihaly. *Flow: The psychology of optimal experience*, volume 41. HarperPerennial New York, 1991.

[3] W. Dann, D. Cosgrove, D. Slater, D. Culyba, and S. Cooper. Mediated transfer: Alice 3 to java. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 141–146. ACM, 2012.

[4] S. Engeser and F. Rheinberg. Flow, performance and moderators of challenge-skill balance. *Motivation and Emotion*, 32(3):158–172, 2008.

[5] L. Grandell, M. Peltomäki, R.-J. Back, and T. Salakoski. Why complicate things?: introducing programming in high school using python. In *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52*, pages 71–80. Australian Computer Society, Inc., 2006.

[6] C. h Pro. Scratch: Programming for all. *Communications of the ACM*, 52(11), 2009.

[7] P. Henriksen and M. Kölling. Greenfoot: combining object visualisation with interaction. In *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, pages 73–82. ACM, 2004.

[8] M. Kölling. The greenfoot programming environment. *ACM Transactions on Computing Education (TOCE)*, 10(4):14, 2010.

[9] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond. The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, 10(4):16, 2010.

[10] B. Moskal, D. Lurie, and S. Cooper. Evaluating the effectiveness of a new instructional approach. *ACM SIGCSE Bulletin*, 36(1):75–79, 2004.

[11] K. Rapeepisarn, K. W. Wong, C. C. Fung, and M. S. Khine. The relationship between game genres, learning techniques and learning styles in educational computer games. In *Technologies for E-Learning and Digital Entertainment*, pages 497–508. Springer, 2008.