UNIVERSITY OF CAPE TOWN

DEPARTMENT OF COMPUTER SCIENCE

CS/IT Honours Final Paper 2019

Title: Three Visual Approaches to Aid the Teaching of Recursion: Visual Coding

Author: Moegamat Ra-eez Stenekamp

Project Abbreviation: BBRV

Supervisor(s): Jecton Anyango, Hussein Suleman

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

Three Visual Approaches to Aid the Teaching of Recursion: Visual Coding

Moegamat Ra-eez Stenekamp
Department of Computer Science
University of Cape Town
stnmoe001@myuct.ac.za

ABSTRACT

Recursion is one of the top concepts that students studying Computer Science should know, however a large portion struggle to grasp it when being taught it. This is due to issues in creating a mental model from the conceptual model that is taught in the class environment. Serious games are found to aid in bridging the gap in the creation of the mental model. The aim of this software development project is to create three visual game-like approaches to aid the teaching of recursion. The approach of interest in this paper is a visual coding user interface. A visual coding user interface is a good approach in aiding the teaching of recursion as it allows for the user to focus on understanding the problem rather than the syntax of its traditional text-based coding counterpart. Most of the users that tested the system developed in this software development project found this form of coding to be more engaging and an easier alternative that allowed their visualization of recursion for the problem to be easier.

1. INTRODUCTION

Recursion is a fundamental concept to understand in programming and Computer Science, especially when it is used as the conceptual model to solve abstract problems [1]. It is also necessary to know recursion in order to understand more advanced topics that are taught later in Computer Science [2]. Even though it is so important in Computer Science, many students struggle to understand this concept [10], with teachers finding it a difficult topic to teach as well [3].

The current conceptual models used in the teaching of recursion are inadequate in creating mental models of recursion for many students learning this topic [4,5]. It has been found that games can help create a mental model due to their visual and interactive nature [6]. Games are therefore a dynamic medium that can be used to bridge the gap between conceptual and mental models.

1.1 Project Aims

The aim of this project is to create three different game-styled approaches to aid the teaching of recursion for first year Computer Science students. These approaches are a visual coding user interface, visual simulation and stack visualization. This project will focus on the visual coding user interface.

A visual coding interface is an alternative form of coding to the traditional text-based user interface. This uses drag and drop mechanics, where the user will be able to select an instruction from a list of instructions shown on the user interface. The user can then click and drag the selected instruction and place it inside scaffolding code, which is code with gaps in it where the user will have to input their chosen instructions from the drag and drop. This will complete the scaffolding code and solve the given problem, such as navigating a character from one side of a maze-like puzzle to another. The user would then click a proceed button which would execute the filled in scaffolding code, where the visualization would run their solution, whether it ends up correct or not, allowing them to adjust their changes until the problem is solved.

1.2 Target Users

This system is intended to be used by first year Computer Science students as they would be learning recursion at this stage of their Computer Science careers. The users who tested this system were first year Computer Science students at the University of Cape Town.

1.3 Report Layout

Related work will be looked at in the next section of this report. This will be followed by the software processes used in the development of this system. Requirement analysis will be shown next, followed by the design of the system, the testing, results and finally the conclusion of this software development project.

2. RELATED WORKS

2.1 Recursion

Recursion is a hard topic for introductory programmers to understand, as Milne and Rowe [11] tested. This test compared the difficulty that students found when learning recursion in their first year of study, with it being in the top six most challenging topics. Even though it is challenging, it is still necessary to know as a Computer Science student, especially since it can be used as a conceptual model that solves many abstract problems [12]. The conversion of recursion as a conceptual model to a mental model seems to be the reason why so many students struggle with the learning of recursion [4,5]. Serious games can create good mental

models with their visual and interactive aspects, making them suitable to helping to create these mental models [6].

2.2 Visual Coding Games

Kazimoglu et al. created a game called *Program Your Robot* [7], which is designed to help students learn introductory programming constructs. This is achieved by giving the users an environment where they can build critical Computer Thinking skills. This environment consists of a drag and drop interface and a visualization. The user creates an algorithm that then is simulated by a character in a mini world, where they will have to complete certain tasks in order to progress onto the next level of the game. Figure 1 shows the design of the drag and drop interface placed around the problem visualization.

Figure 1 *Program Your Robot* [7].

Yaroslavski [8] created a phone app called *Lightbot*. This is a serious game in which you must make a robot walk across maze-like levels, accomplishing various tasks of progressively increasing difficulty. Figure 2 shows *Lightbot* and its drag and drop form of coding input, which the user uses to make the character finish the level's challenges and progress in the game. The aim of this game was to teach players programming constructs.

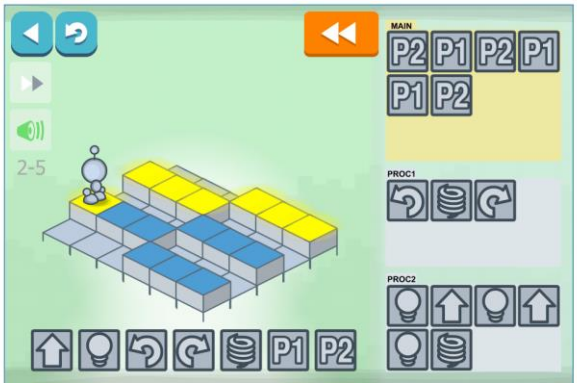

Figure 2 *Lightbot* [8].

Tessler, Beth and Lin [9] created a mobile game called *Cargo-Bot,* which was intended to help students learn recursion via having them teach a robot how to move crates. Figure 3 shows that this game works by having the user create recursive functions out of basic commands such as movement directions and the function names that would have to be used in the functions themselves to be recursively called. Recursion can be broken down into two core parts, the active flow and the passive flow. The active flow is the forward passing of control where the stack gets called by the program. The passive flow is the backwards flow of control, in which control is passed back once the active flow has filled up all the calls it can on the stack [13]. This game shows the active flow of recursion; however, the passive flow of recursion cannot be viewed.

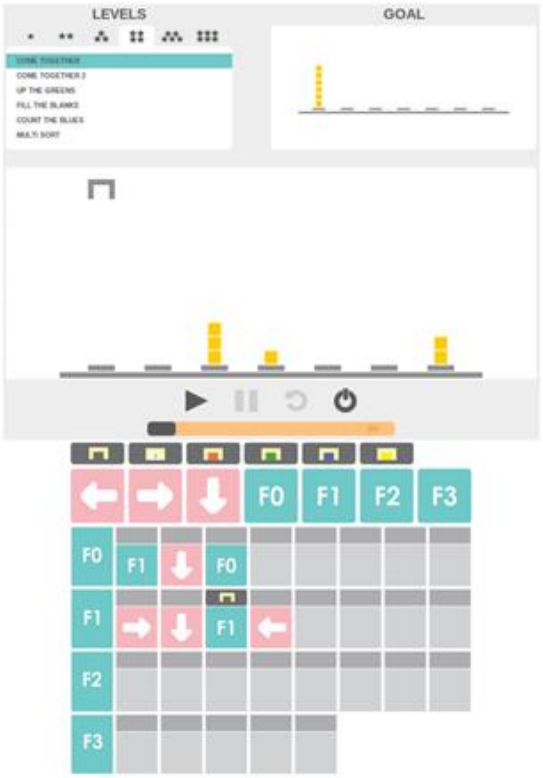

Figure 3 *Cargo-Bot* [9].

3. SOFTWARE PROCESS

The chosen software process was an iterative agile approach. This was chosen over other forms of software processes to ensure that this system will achieve its overall goal of helping students learn recursion. This also guarantees the system caters to the user's needs and what would benefit them as opposed to what we speculate would benefit them. This would be achieved by using this software development process as the system would be flexible and able to change as the user's requirements are changed and updated (see user requirements below).

This visual coding user interface was developed in 5 core iterations, which are described in detail in the design process.

3.1 Software Specifications

This system was created using Unity, which is a game developing engine. The code for this system was created using the programming language C#. The programming language Python was chosen as the language of choice for the users to interact with via the scaffolding code. Python was chosen as our target users who tested this system had been taught Python.

4. REQUIREMENTS ANALYSIS

Requirement gathering is crucial for the creation of this project. This ensures that the system will cater to the needs of the users.

4.1 Requirements Gathering

Since this system is designed to be used with first year Computer Science students, it was decided that before construction of this system, 10 first year Computer Science Students would be interviewed. This was done by getting the correct ethical clearance to be allowed to interview the students. The target interviewees were determined to be first year Computer Science students who were studying CS1016S in the second semester of 2019. This group of students was chosen for feedback as they would have recently finished learning recursion from the prior course CS1015F, allowing their opinions, findings and struggles of learning recursion to still be relatively new. In order to interview these students, it was arranged with the course convener of CS1016S to allow for the students to be interviewed, with their consent, during one of their Friday lab sessions.

They were asked several questions regarding a visual coding user interface and recursion (full notes that were taken during the interviews are available on the website of this project [14]). They were asked what they found difficult about recursion. This was enquired in order to see what aspects of learning recursion caused them to struggle, so that this system could help address these issues.

Most of the students said that the reasons they found recursion hard was because of the way it was taught and that there was not enough time to learn such a complex topic. Another key thing that many of the students said was that it was hard to visualize the concept of recursion and how it works. Less popular, but still notable answers include: recursion is not as intuitive as loops and that there wasn't anything in their lives that they could relate to recursion, so it was hard to relate to, making it tougher to understand.

Regarding the visual coding user interface, the students were asked two questions: if they would consider a drag and drop visual coding user interface less daunting/ intimidating than regular text-based coding and why they thought it would be more/ less intimidating and finally whether they thought that recursion should be first coded using a visual interface instead of a text-based interface. These questions were asked so whether the students would find the visual coding user interface helpful could be gauged.

60% of the students said that they would prefer visual coding, with only 30% of the students preferring text-based coding and the remaining 10% of the students finding that they were equivalent.

90% of the students thought that there should be a transition from visual coding to text-based coding, with the remaining 10% of the students being against visual coding entirely.

From these results we can see that the students struggle a lot with recursion because of the time they have to learn it, as well as their lack of ability to visualize recursion easily. It is also clear that the majority of the students want a form of visual coding that transitions into text-based coding.

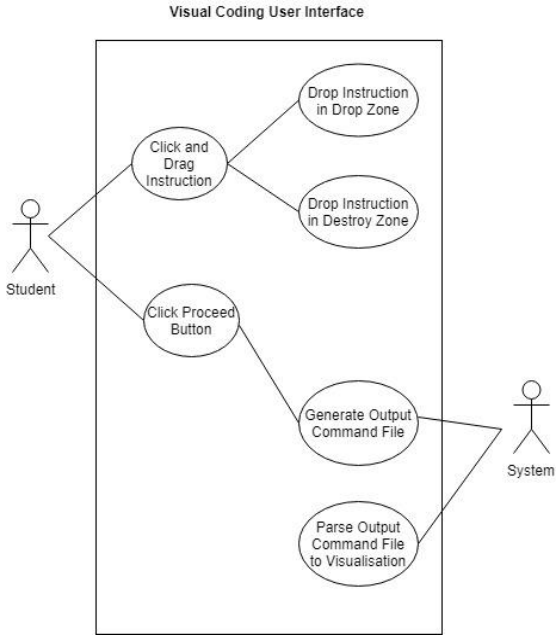

Figure 4 Use case diagram of visual coding user interface.

4.2 User Requirements

The requirements gathering process, alongside the use case diagram for the visual coding user interface shown in Figure 4 (all UML diagrams used in this report and a few additional ones are available on this project's website [14]), where used to determine the user requirements.

It was determined that the system had to be developed in a way that would be intuitive for the users to use, and that would help develop a visualization of recursion. It is important to represent the problem in a way that would prevent them from thinking about solving the problem using loops, so that they could focus on their visualization of recursion.

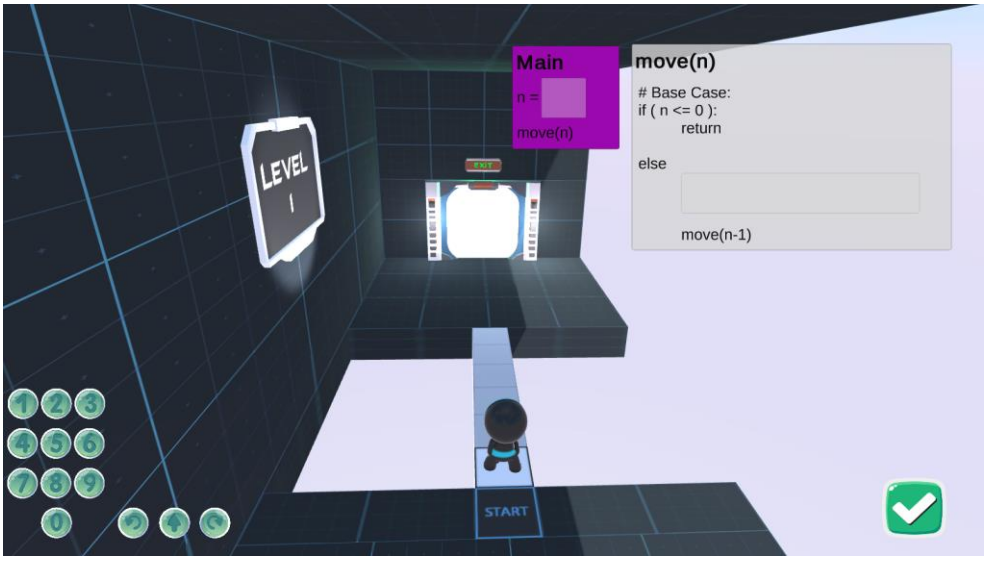

Figure 5 Visual coding user interface displayed above the visual simulation for the first level.

This system would have to work fast, smoothly and seamlessly. The visualization should run immediately after the proceed button is pressed once the scaffolding code has been filled in.

5. DESIGN

A description of the high-level design of this system will be given, giving a deeper understanding of this system. The details of design will then be shown, in which the system, and how it works will be gone through in detail, followed by the class interaction and finally the design process gone through in the creation of this project.

5.1 High-Level Design

This system was created on Unity within the canvas game object, which is the user interface element available in Unity. The user interface layout and design can be seen in Figure 5. This system has 7 core functions that work together in order to make the visual coding user interface work (these core functions are explained in detail below). The 'question problem' is the actual coding problem that is represented in the form of scaffolding code, with gaps where 'drop zones' are, in which the user will complete the scaffolding code. 'Instructions' are the "answers" the student will have to drag in order to complete the scaffolding code given for the question problem. An 'instruction zone' holds all the instructions for the level. 'Drop zones' are the areas in which the user will be able to drop their instructions into the scaffolding code. These restrict the amount of instructions in them depending on its size, ensuring that the problem has to be solved recursively. The 'destroy zone' is everywhere on the screen other than the instruction zone and the scaffolding code with the drop zones on them. Unwanted instructions can be dropped in the destroy zone to delete them. The 'proceed button' will run their solution once it is pressed. The 'level manager' controls the flow of the levels and allows for the user to switch between levels at the start of the game as well as when progress onto further levels.

5.2 Details of Design

5.2.1 Question Problem

This is an area on the canvas filled with text. This text is the recursive scaffolding code, which helps guide the user's solution. There are gaps in the text, which are "holes" in the scaffolding code. These gaps have drop zones in them, allowing the user to complete the scaffolding code with the given instructions. The question problems and their drop zones change depending on the level that the user is on. The arrow with the label 1 in Figure 6 shows the question zones for the 'Main' and 'move(n)' functions of the first level.

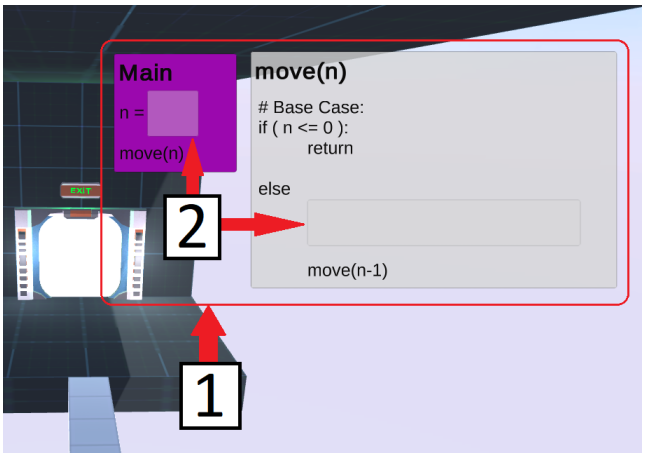

Figure 6 Drop Zones for the Main and move(n) functions of level 1.

5.2.2 Instructions

Instructions are the potential answers which the user must choose between, after which they will drag and drop the answers onto the correct zones in order to complete their solution to the recursive coding problem. Some of these instructions include movement commands and number variables.

Each instruction is a game object with a script called "Draggable" attached onto it. This script takes the mouse input, and when the mouse clicks and drags, if it is over an instruction, then a copy of the instruction is made if it is in an instruction zone, while the original instruction is dragged instead if it is clicked from the drop zone. This instruction is then moved with the mouse. When the click and hold is released, the instruction has various interactions, depending on the zone it is dropped in. When an instruction is dropped in the destroy zone, the instruction is destroyed, however, when it is dropping in a drop zone, it remains there.

Figure 7 shows the layout of the instructions, with the arrow labelled 1 pointing to a single instruction. Figure 8 shows sample instructions after they have been dragged and dropped in various drop zones.

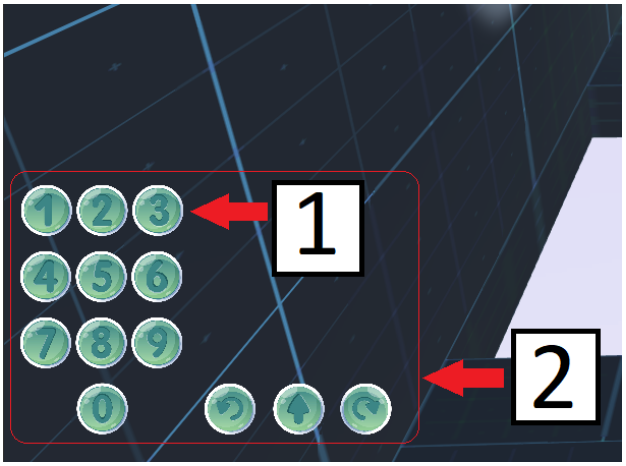

Figure 7 Instructions layout in the instruction zone.

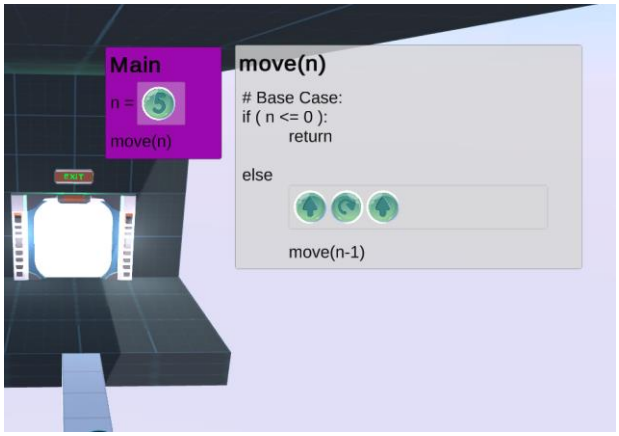

Figure 8 Instructions dropped in drop zones.

5.2.3 Instruction Zone

The initial instructions the user must choose from are stored in this zone. The instructions here do not move when they are dragged, but rather copies are created, and those copies are moved instead. This allows the user to use the same instruction multiple times. The arrow labelled 2 in Figure 7 shows the instruction zone and its instructions (in this figure, the instructions includes the numbers 0-9, forward, and left and right turn instructions).

5.2.4 Drop Zone

This is the areas in the scaffolding code where the user must place their chosen instructions that completes the scaffolding code and answer the coding problem. The sizes of these drop zones affect how many instructions can be placed in a drop, ensuring that the solutions can only be solved recursively. The arrows labelled 2 in Figure 6 shows examples of drop zones.

Each of these zones has the "DropZone" script attached to it. This script takes in the position of the mouse's pointer and checks if the mouse is holding an instruction. If the instruction is being held over a drop zone, the user can move the instruction in between the other instructions in that drop zone depending on where it is hovering over and if there is still space in the drop zone. This allows the user to rearrange instructions in the drop zones and place them anywhere in the list of instructions they put there. This is done by creating a placeholder object that is empty/ has no image, but it is in the dimensions of an instruction.

5.2.5 Destroy Zone

This zone is all the areas of the screen other than the question problem, with its drop zones, and the instruction zone. When an instruction is dropped onto this zone, it is destroyed.

5.2.6 Proceed Button

The proceed button is represented as a big tick in the bottom right of the screen. The purpose of this button is for the user to press it once they have finished filling in the scaffolding code with the appropriate answers. This then generates a .txt file with a series of movement commands created using the user's solution. If any of the scaffolding code has incomplete or incompatible instructions, then a message is displayed for the user to use to alter their solution. This .txt file is read by the visual simulation approach of this project, which then simulates and tests the user's solution. The arrow labelled 1 in Figure 9 shows the proceed button. This button is controlled by the "OutputGenerator" script.

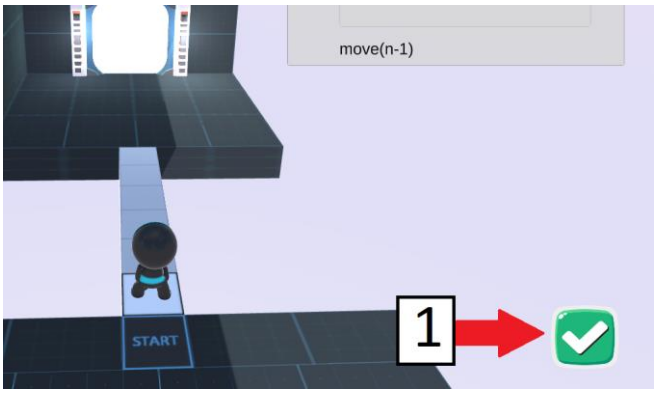

Figure 9 Proceed button.

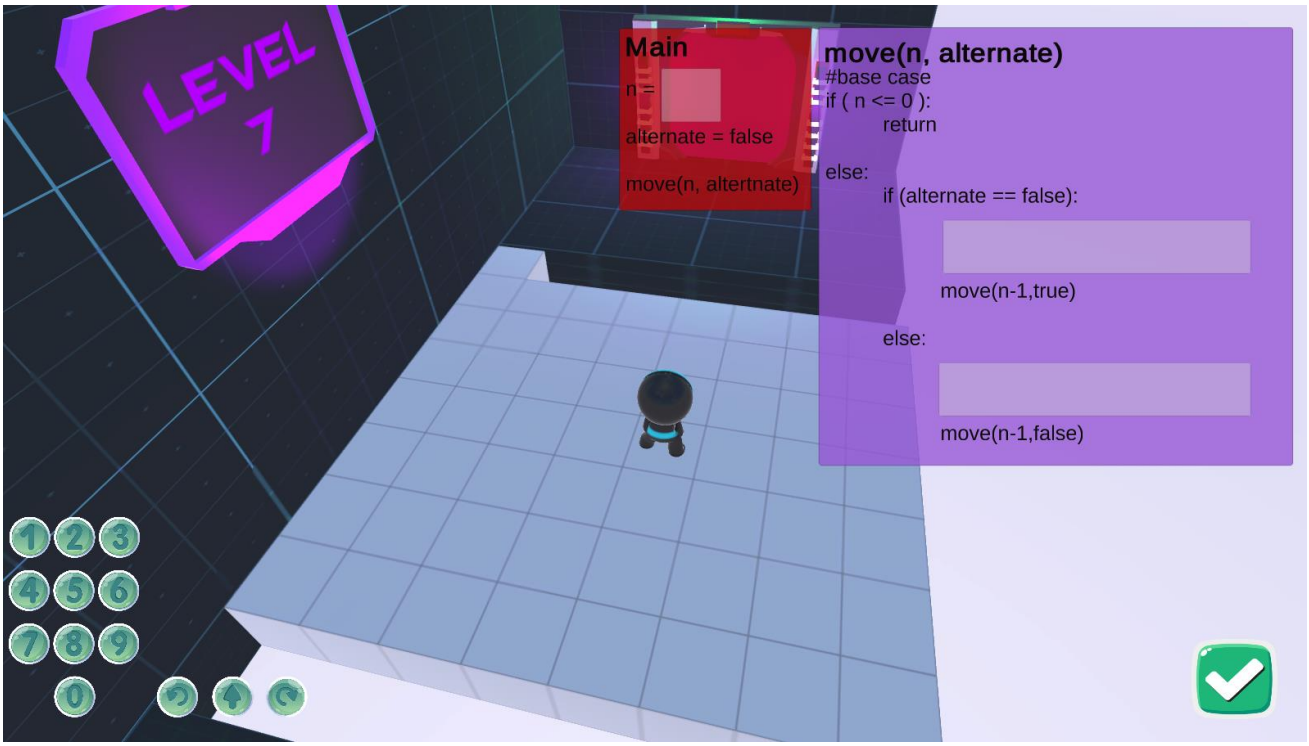

Figure 10 The visual coding user interface for the seventh, and final level.

5.2.7 Levels

There are seven levels used in this game. They are basic navigation maze-like problems in which the user will have to navigate from one end to the other by filling in recursive code. These problems range in difficulty, with it getting increasingly difficult with each level. Level 1, which is the easiest level, can be seen in Figure 5. The most complex level, level 8, can be seen in Figure 10. For the full list of all the levels, refer to this project's website [14]. These levels are designed in a way in which they can only be solved recursively due to restrictions in the drop zone sizes.

5.2.8 Level Management

This serious game used multiple levels of increasing difficulty. This requires a level management system, which keeps track of the different levels, as well as giving an option to choose which level you want to play. This allows more advanced users to skip ahead in levels.

The levels are managed via two scripts, the "ModeSwitcher" and "Level Manager" scripts. The "ModeSwitcher" script has the function to load between given levels, depending on the interaction in the game. This will be called when proceeding to the next level as well as when selecting levels. The "LevelManager" script updates the "ModeSwitcher" script with the right values for a level, and then calls the "ModeSwitcher" method to load in that level.

5.2.9 Integration

The visual coding user interface must only be integrated with the visual simulation part of this project as it was decided from the user requirements that the system should transition from visual coding to text only coding, which is how the stack simulation gets its input. The way that this is integrated is by having the visual coding user interface displayed on the screen in front of the visual simulation. There is a free-roam state, in which the user can free move the character, however once they walk on the start block, they will the coding mode, displaying the visual coding user interface.

5.2.10 Assets

All assets used are picked in theme with this serious game and are free to use and royalty free.

5.3 Class Interaction

Figure 11 shows the class diagram of the visual coding user interface, where the circles represent various game objects in the project.

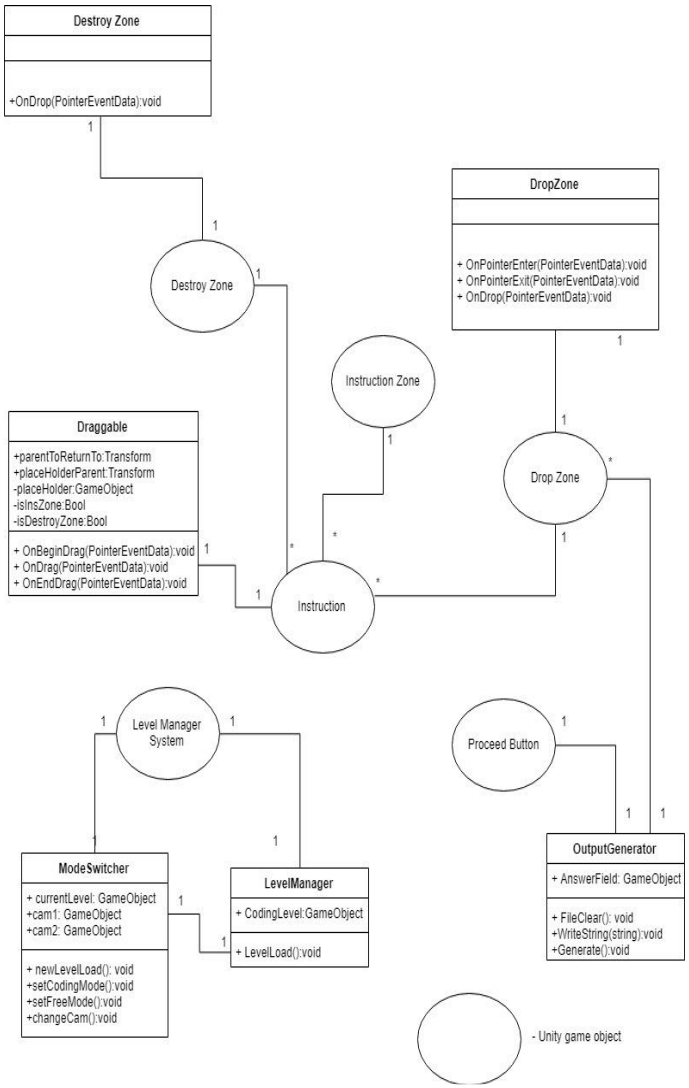

Figure 11 UML class diagram of visual coding user interface.

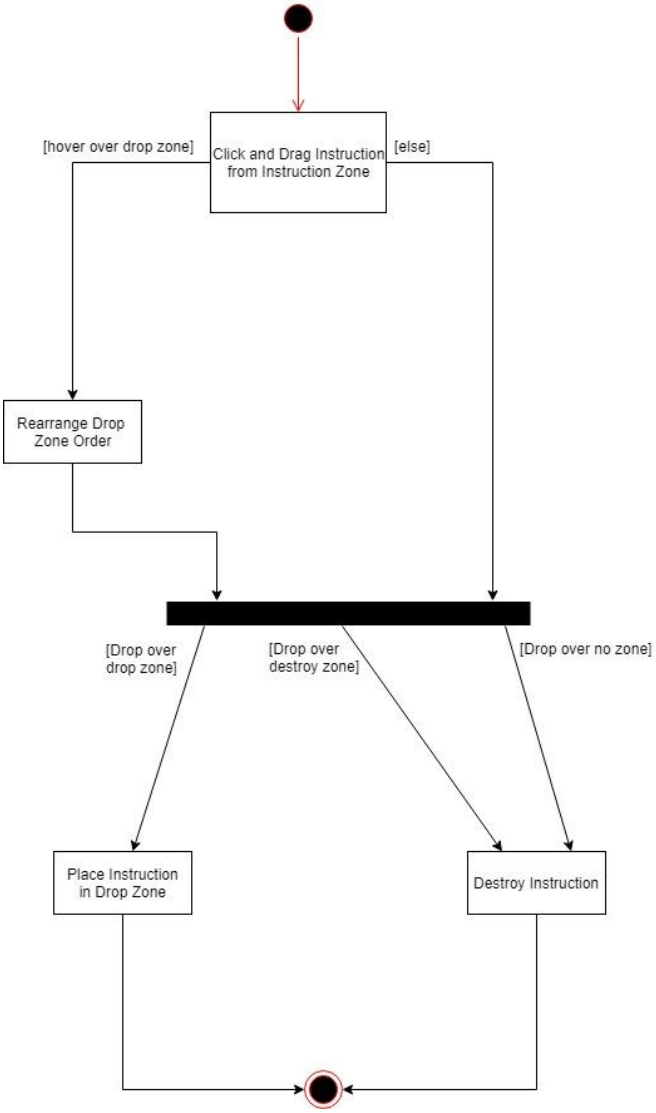

Figure 12 Activity diagram of an instruction dragged from the instruction zone and dropped in various zones.

Figure 12 shows an activity Diagram of when an instruction gets dragged from, the instruction zone and dropped on the various zones.

5.4 Design Process

The visual coding user interface was developed in five core iterations: early envisioning, increased specification, output and assets, integration and user feedback.

5.4.1 Iteration 1: Early Envisioning

In this iteration we discussed our ideas for what we thought the users would want out of our system. We then created paper prototypes of what we thought the system layout would look like and how the users would interact with them. A very rough version of the system was created on Unity with these specifications. The build at this stage included the instructions dragging from an instruction zone into basic drop and destroy zones.

5.4.2 Iteration 2: Increased Specifications

Here we arranged with the course convener for the CS1016 Computer Science course at UCT to hold interviews with some of their students to ask about what they would want out of this project (see requirements specifications above).

These new specifications from the users were taken as our focus, so we tailored the build from iteration 1 as well as what was still to be made further according to these new user requirements.

In this iteration, the instructions, instruction zone and drop zones had the more complicated functionality described above

implemented. Tweaks to this functionality and layout were made according to the new user requirements. The scripts involved with level switching were also created here

5.4.3 Iteration 3: Output and Assets

In this iteration, since the functionality of the user interface instructions were working correctly, recursive problems were decided on in this iteration. The proceed button was created as well as the basic text output that would be used later with the Visual Simulation.

The assets that would be used were chosen during this iteration. These assets must be royalty free as well as free of charge. These assets had to be initiative for users to use as well as fitting in with the overall theme of the project that is shared among all the parts.

5.4.4 Iteration 4: Integration

In this iteration the focus was on getting the visual coding user interface to work with the visual simulation section, including the level switching and when the visual coding user interface would appear and disappear in the game.

5.4.5 Iteration 5: User Feedback

In this iteration user testing occurred. The system requirements were updated after this, and tweaks to the system according to the user feedback and requirements was added.

6. TESTING AND EVALUATION

We tested the systems we developed in Unity's play mode. We also had students who are the target users of our system evaluate whether our system achieves its goal of aiding the teaching of recursion and its usability.

6.1 Procedure

Extensive playtesting in Unity's play mode allowed us to ensure that there were not any bugs in the system and that all the features and functions worked as intended.

All possible solutions that would solve each level was thought of and tested. This ensured that the only working solutions to each level had to be coded recursively as the drop zone restrictions were adjusted and limited, preventing iterative solutions to the levels.

It was decided to get the first year CS1016S students as was done when getting the user requirements, since this is a user-centered system. To do this, we underwent the same process of contacting the course convener of the course to get permission to send an announcement out to the students requesting for them to sign up for 30-minute evaluation slots.

In the evaluation slots, the students were presented with a laptop, on which they ran our software. After they completed playing the serious game, they were given 3 questionnaires, one for each of the parts of this project respectively.

In order to evaluate the user experience and usability of this system, the Game Experience Questionnaire [15] was used. The Game Experience Questionnaire is a standardized questionnaire that was created to ask the user questions pertaining to their experience when playing a game. These questions are answered on a scale of 1 to 5 with a total of 30 questions. These questions were very general questions that do not refer to the visual coding user interface specifically, but rather the user's experience with playing the game. Examples of these questions are whether they enjoyed playing it and if they found it challenging. The full list of these questions as well as their answers can be found at this project's website [14].

Three additional questions were asked regarding the visual coding user interfacing particularly. These questions included: whether they found the visual coding user interface intuitive and easy to use and what they would change about it; whether they liked the way it looked and why they felt that way; and finally, whether they would have found it easier than solving the problem using a standard text-based alternative with reasoning.

6.2 Results

6.2.1 Game Use Experience

The results of the Game Experience Questionnaire's 30 questions were all constant responses with occasional outliers from the general opinion. The averages of all the results for each major category are shown in Figure 13. All the individual results can be found on this projects website [14]. The general results all point towards a positive game experience. There is one question with notable results however. This was question 19, which asked whether the user was fast at reaching the game's targets. The result of this question had an even amount of people that finished at a fast time as those that found they went at a relatively slow speed, with the majority of the students finding that they took a moderate time, as Figure 14 shows.

Component	**Average Score(see key)**
Competence	3.64
Sensory and Imaginative Immersion	4.11
Flow	3.65
Tension/ Annoyance	1.3
Challenge	2.01
Negative Affect	1.29
Positive Affect	4.52

Key:

1 – not at all 2 – slightly 3 – moderately 4 – fairly 5 – extremely

Figure 13 Game Experience Questionnaire results.

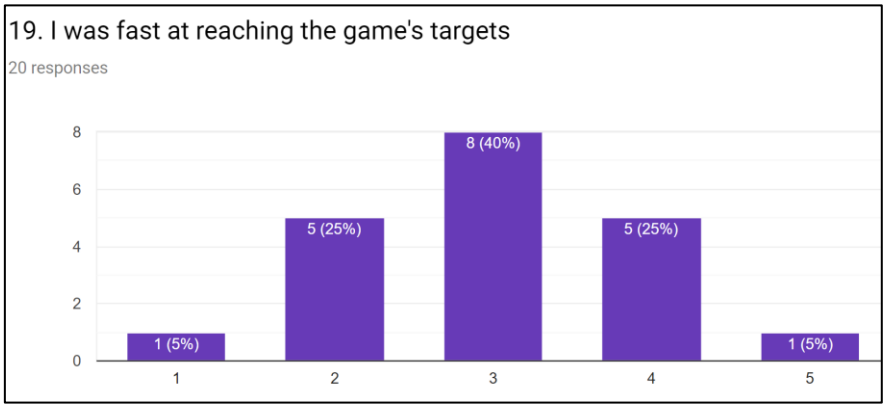

Figure 14 Question 19 Answers from Game Experience.

For the visual coding questions, all the responses can be found on the website [14]. The question about whether they found the visual coding user interface intuitive and easy to use had 95% of the users finding it easy and intuitive. The 5% who answered no to this question said they felt this way as they would have preferred to type the symbols instead of dragging and dropping them. For the second part of this question, which was what they would change about the system, most of the students found that they would not change anything other than adding clearer instructions on its use at the start. This result was taken into consideration after the test and the instructions at the beginning were expanded upon during the 5th development iteration. As for whether they liked the way the visual coding user interface looked, they were all content. For the final question -whether they would find it easier than text-based coding- 85% of the students said that they would have found this easier than text-based coding, with 15% of the students stating that they would have found it harder. For those who found it would be harder than text-based coding, they said it was because the visual coding added to the amount of effort that they would need to solve the problem, with responses saying that they would have found writing code themselves easier to learn and master recursion.

7. CONCLUSION

This project sought to create three different game-styled approaches to aid the teaching of recursion for first year Computer Science students, with the approach of this paper being on a visual coding user interface. This system had to be able to aid the students in their understanding and solving of recursive problems without having to worry about syntax, but rather focusing on understanding and visualizing the problem.

It can be shown from the results that the users found the visual coding user interface better than the current text-based coding alternatives when it comes to visualizing and solving the problem of recursion. The usability of the system is also shown to be effective due to the responses of the user experiences being mostly positive and with the students finding it easy and intuitive to use as well as liking how it looked. The results also show that the increasing difficulty of the levels are of the right level for first year students to learn recursion. Therefore, the system that was developed was a success and will be able to use this program to aid their visualizing and learning of recursion, which will help them in their computer science careers.

8. LIMITATIONS AND FUTURE WORKS

This system was limited to 20 students who evaluated the system. This is an area that could be expanded on in future work on this system. 7 levels were created for this system; however, it has been developed in such a way that future levels could be easily created and added to this system, allowing for versatility for future endeavors.

7. REFERENCES

[1] Wu, C. C., Dale, N. B., & Bethel, L. J. (1998, March). *Conceptual models and cognitive learning styles in teaching recursion*. In ACM SIGCSE Bulletin (Vol. 30, No. 1, pp. 292-296).

[2] Milne, I., & Rowe, G. (2002). *Education and Information Technologies*, 7(1), 55–66. doi:10.1023/a:1015362608943

[3] Eagle, M., & Barnes, T. (2009). *Experimental evaluation of an educational game for improved learning in introductory computing*. ACM SIGCSE Bulletin, 2009, 321-325.

[4] Stasko, J., Badre, A., & Lewis, C. (1993). *Do algorithm animations assist learning?* Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '93. doi:10.1145/169059.169078

[5] Wu, C.-C., Dale, N. B., & Bethel, L. J. (1998). *Conceptual models and cognitive learning styles in teaching recursion*. Proceedings of the TwentyNinth SIGCSE Technical Symposium on Computer Science Education - SIGCSE '98. doi:10.1145/273133.274315

[6] Kirchgessner, M., & Ketelhut, D. J. (2012). *Video games and learning: Teaching and participatory culture in the digital age.* Science Education, 96(5), 963–965. doi:10.1002/sce.21020

[7] Milne, I., & Rowe, G. (2002). *Education and Information Technologies*, 7(1), 55–66. doi:10.1023/a:1015362608943

[8] Gouws, L. A., Bradshaw, K., & Wentworth, P. (2013, July). *Computational thinking in educational activities: an evaluation of the educational game light-bot.* In Proceedings of the 18th ACM conference on Innovation and technology in computer science education (pp. 10-15).

[9] Chaffin, A., Doran, K., Hicks, D., & Barnes, T. (2009). *Experimental Evaluation of Teaching Recursion in a Video Game* (pp. 79-86).

[10] Milne, I., & Rowe, G. (2002). Education and Information Technologies, 7(1),

55–66. doi:10.1023/a:1015362608943

[11] Milne, I., & Rowe, G. (2002). *Education and Information Technologies*, 7(1), 55–66. doi:10.1023/a:1015362608943

[12] Wu, C. C., Dale, N. B., & Bethel, L. J. (1998, March). *Conceptual models and cognitive learning styles in teaching recursion*. In ACM SIGCSE Bulletin (Vol. 30, No. 1, pp. 292-296).

[13] Götschi, T., Sanders, I., & Galpin, V. (2003). *Mental models of recursion* (Vol. 35, No. 1, pp. 346-350).

[14] Honours Project Website: http://projects.cs.uct.ac.za/honsproj/cgi-bin/view/2019/tony_shak_raeez.zip/csa/index.html

[15] IJsselsteijn, W. A., De Kort, Y. A. W., & Poels, K. (2013). *The game*

experience questionnaire. Eindhoven: Technische Universiteit Eindhoven.