Three Visual Approaches to Aid the Teaching of Recursion

Project Proposal

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

Heng Jia (Tony) Guo
Computer Science
University of Cape Town
South Africa
gxxhen001@myuct.ac.za

Shakeel Mohamed
Computer Science
University of Cape Town
South Africa
mhmsha056@myuct.ac.za

1 PROJECT DESCRIPTION

Recursion is one of the most important programming concepts in computer science. It can enable the creation of very simple algorithmic solutions to certain problems that would otherwise be unsolvable or inefficient with any other type of approach. It is a fundamental concept in computer science, whether understood as a mathematical concept or programming technique [1]. It is, however, regarded as a challenging topic to learn for students being introduced into the world of computer science [7]. Educators often find it a difficult topic to teach as well [2]. Many students fail to grasp the concept as it is taught in lectures and textbooks. Students may find it difficult to cope with more advanced topics taught later in their CS courses if they have not fully understood recursion as the concept can be applied to many other areas [7].

The lack of student understanding of recursion can be attributed to the current conceptual models of recursion taught, being hard to transfer into mental models for the students [6,8]. A dynamic medium is needed to bridge this gap between conceptual and mental models. Games can be used as a medium that achieves this due to their visual and interactive aspects [5].

This project will explore three different game-inspired approaches to teaching recursion. These are intended to be used in the form of assignments that are given to the students. These approaches will make use of various elements from games, particularly simulation, visualization and animation. This is designed to replace or supplement a standard coding assignment approach.

2 PROBLEM STATEMENT

2.1 Aims

This project will focus on the evaluation of the user experiences of three different game-inspired approaches used to teach recursion. The approaches that will be evaluated are: visual coding, visual simulation; and stack visualization. With this evaluation, key characteristics of the different game-style approaches to teach recursion will be identified.

This project will provide an alternative form of environment as opposed to the typical text-only environment students are used to. These alternative environments will be individually evaluated in order to determine the user experience. The environments are designed to help students better visualize and understand the concept of recursion in an ongoing and interactive way, while still retaining the technical, real-life aspects of programming.

2.2 Research Questions

1. What is the user experience of the visual coding approach used to teach recursion?
2. What is the user experience of the visual simulation approach used to teach recursion?
3. What is the user experience of the stack visualization approach used to teach recursion?

User experience is evaluated instead of learning since learning takes a long period of time to evaluate and there are a large number of confounding factors, such as prior experience.

3 PROCEDURES AND METHODS

The development of the three educational software tools will be for desktop use only in the form of a browser application. This project encapsulates the development of three different game-based software tools that include the following three approaches to teaching recursion: visual based, visual simulation and stack visualization.

There will be two different input forms: text based (they will be given scaffolding code as a base on which their solution can be built) and visual based (a drag and drop interface). There will be two output forms: visualization of the stack, and visualization of the execution of the code. The visualization of the stack and the visualization of the execution of the code will make use of both text and visual based input (see Figure 1). The programming language in which the user will be required to code in for the textual coding approach is Python. However, this will be a subset of python which will be the interface among modules.

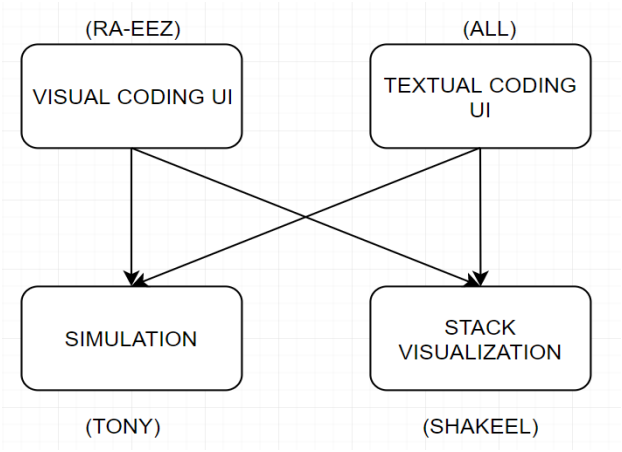


Figure 1: Work Allocation

The user experience for each of these approaches will be assessed to identify the key user experience characteristics of each of these approaches. Each of these approaches will have their user experience tested using similar recursive problems. We will make use of IJsselteijn et al.'s game experience questionnaire as the main assessment instrument in evaluating the user experience for each of the three approaches [13]. We aim to evaluate using a sample size of 20 first year students, as they would have just started learning the topic.

The sections below will examine the procedures and methods that will be used to answer the research questions outlined in Section 2.

3.1 Visual Coding

This involves a visual interface the student must use to construct an algorithm that will solve a recursive problem, such as guiding a character through a maze or path. The interface will be a 'drag and drop'' type with movement command blocks that can be placed in a main 'program block', which will then be executed upon user compiling the code. This approach is more visual in nature and requires little to no syntax knowledge. It provides a more game like experience relative to the others due to the interactive nature of visual coding. Thus, this can be more appealing to students. More emphasis is placed on recursive thinking rather than written syntax.

An evaluation will be conducted on whether this visual approach is effective in helping students have a better experience in learning how recursion can be used to solve problems when syntax does not need to be considered. The users' experience with a drag and drop coding user interface will be taken into account.

3.2 Visual Simulation

When code is run, a visualization of their coded solution to the problem will be displayed. This will be in the form of a character performing tasks relating to the execution of the code within a maze-like puzzle environment, similar to that of Program your robot [10]. This method is a good way of learning as it allows for the students to try solving a problem using recursion as well as seeing the effects of their code on the problem. The end visual simulation acts as an incentive and will motivate students to solve the recursive problem. We hypothesize that this will help students understand recursion as they can see the results due to their own implementation of recursion.

The evaluation of the user experience will note how well they understood the visualization of their solution and whether they felt it was an accurate representation of their solution.

3.3 Stack Visualization

The call stack keeps track of function calls. It is the list of all the functions currently running at any given point in the program. Any time a call stack hits a return, it pops the current function off the stack and goes back to whichever function is now on top. This information, during the execution of a recursive program, can be found in most IDE's, however it can be difficult to conceptualize without it's visual representation.

This approach demonstrates how the call stack grows and shrinks as a recursive program runs. In doing so, this gives the student a clearer understanding of what is happening 'behind the scenes' of their recursive code. The user will complete a given segment of code and, once compiled, a visualization will appear to demonstrate their programs' call stack.

This will help students to understand the recursive flow of control, which is essential in understanding recursion. The recursive flow of control consists of two parts: the active and passive flow. The active flow is how each recursive call will lead into the next recursive call until it hits its base case. The passive flow is the backpropagation of these calls that returns a final answer.

The evaluation of the user experience with stack visualization will include checking whether users found it helpful in their understanding of recursion and the recursive flow of control through the visualization of the stack.

4 ETHICAL, PROFESSIONAL AND LEGAL ISSUES

Ethical issues are identified in the testing and the actual software we develop. Prior to testing with users, ethical clearance will have to be obtained from the UCT Human Research Ethics Committee. This is done by submitting an ethical admittance form to the committee. Our final products and report will belong to the developers. No personal information will be collected from users testing our software and their permission to use their feedback will have to be obtained. This research will not be conducted in the classroom; hence it will have no negative effects on the student's learning as it will not replace any material taught in the curriculum. The intellectual property of the project will belong to Shakeel Mohamed, Moegamat Ra-eez Stenekamp, Tony Guo and the University of Cape Town. The research paper of the project will be free and open for distribution and future research.

5 RELATED WORK

There have been several different approaches to attempt to gamify and visualize the concept of recursion. This section will take a look at different works that have utilized these different approaches.

5.1 Visual Coding Games

Tessler, Beth, and Lin rewrote the game *Cargo-Bot* to teach recursion to students by making students teach a robot how to move crates [3]. This mobile game uses movement blocks instead of code to design the program that manoeuvers the cargo crates. In addition to being addictive and fun, it motivates the students to explore the game and its mechanics. However, this game does not provide a fully conceptual model of recursion. Figure 2 displays one's ability to play the simulation representing the active flow. On the other hand, the passive flow of the recursion is not visibly shown and cannot be traced with the game. Hence, the concept of recursive calls popping from the stack cannot be easily grasped.

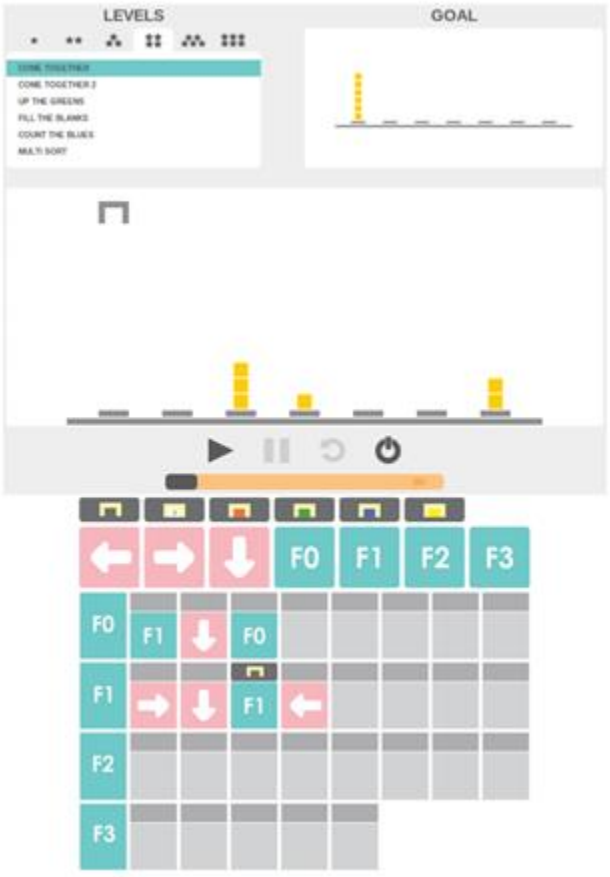


Figure 2: Gameplay in Cargo-Bot [3]

In Dann et al.'s paper [9], a 3D animation world builder was used to create visualizations for the recursive solutions of problems. The game depicts Alice doing repeated actions that get closer and closer to completing a task. Some of these tasks were Rabbit and Butterfly Chase (having a rabbit chase a butterfly) and completing the Towers of Hanoi illustrated in Figures 3 and 4. This approach received a high level of student involvement and the ability to develop an intuitive understanding of recursion through visual feedback. However, it was questioned whether this approach taught recursion fully as the passive flow of recursion was not shown in the animations. The animations only showed the active flow, similar to that of the process of iteration, missing the passive flow that encapsulates the full recursive flow of control.

Figure 3: Rabbit and Butterfly Chase [9]

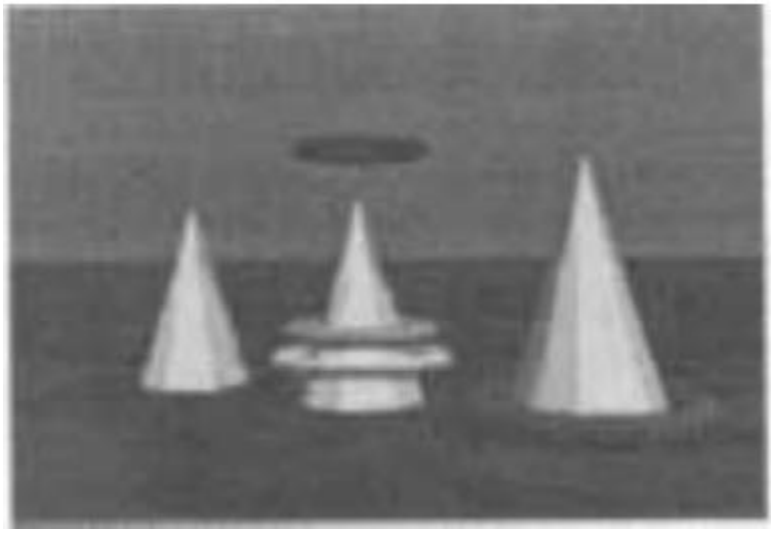


Figure 4: Towers of Hanoi [9]

Kazimoglu et al. developed the game *Program your robot* [10]. It is a serious game designed to help students learn introductory

programming constructs by enabling them to practice working within an environment that explicitly supports the acquisition of Computer Thinking skills (such as algorithm building, debugging, and simulation).

The game uses drag and drop mechanics to form an algorithm that is simulated by a robot in a block-like environment. The level is passed if the commands in the algorithm make the robot do the required objectives. Figure 5 shows the pleasing visuals and achievements in the game that keep students engaged and motivated. Through the ease and power of creating an algorithm by dragging and dropping commands, students can enhance their problem-solving skills and gain an intuition for programming.

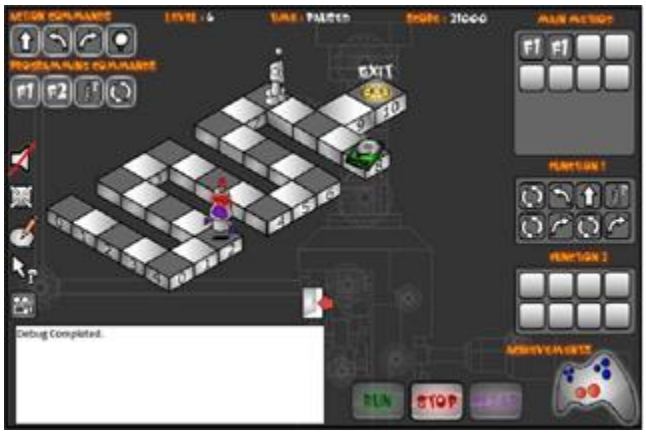


Figure 5: Level 6 in Program your robot [10]

5.2 Visual Simulation Games

Chaffin et al. present the game EleMental: The Recurrence [4]. This game focuses on writing code and allows programmers to interact with the game through a depth-first search of a binary tree to come up with the solution to the coding question. Figure 6 introduces a game world where there are many things to learn and explore that immerses the students in a deep approach to learning. A visual representation of their solution is also played. Results from students showed that the visualization of recursion was the favorite aspect whereas the gameplay was the least favorite aspect. This game only taught a specific case of recursion using depth-first search; a wider range of cases are needed to fully encapsulate the concept.

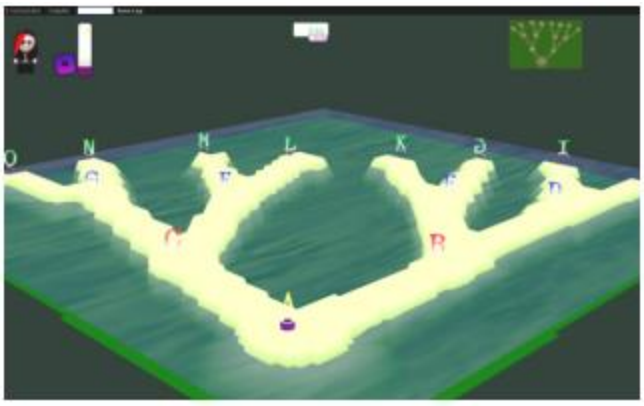


Figure 6: Level 2 AI walkthrough [4]

5.3 Stack Visualization Games

Elenbogen and O'Kennon use Turbo Prolog, a programming language, to demonstrate recursion graphically using fractals [11]. Due to Prolog's development environment, it makes the recursive flow of control transparent and easy to follow. With the use of fractals as the resultant figures of programs, it is particularly easy to analyze the fractals with their recursive nature. Mayer and Gallini conclude that an illustration is most valuable when the illustration explains the concept and when the student lacks previous experience [18]. The use of fractals can expose the learner to a new experience of recursion. By making use of a stack in place of fractals, the visualization of the recursive flow of control can be enhanced, aiding students in their understanding of the concept.

Leroux et al. had an interesting take on stack visualization and introduced the visualization tool Jacot [12]. This tool creates two different visualizations for the execution of concurrent Java programs. These visualizations assist the user in understanding concurrency concepts such as synchronization, non-determinism and deadlock. Figure 7 shows these visualizations. Although these visualizations are not that abstracted and do not include much imagery, they still assisted the user in understanding new concepts.

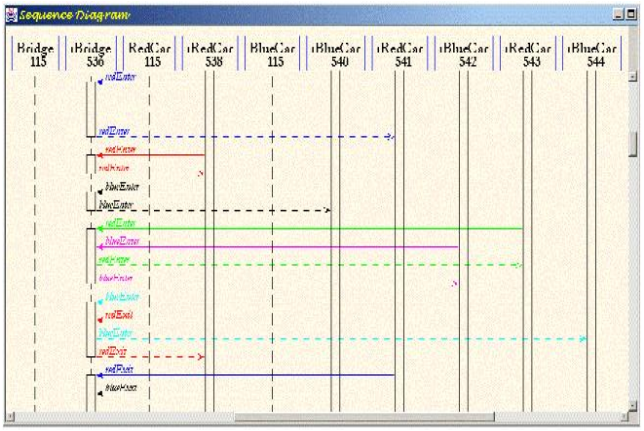


Figure 7: One visualization view of a concurrency program [12]

6 ANTICIPATED OUTCOMES

This section will examine the anticipated outcomes of each of the three approaches.

6.1 Visual Coding

The features of the visual coding environment will include a GUI with selectable program 'blocks', which perform different functions in order to move a character around a 3D environment (such as a maze or path). These blocks will have movement functions such as up, down, left, right and rotate. These blocks can be dragged and dropped into a main program. Once a series of movement command blocks have been created, which should be the solution to the maze/path and have been built and compiled, a visual execution of the character 'acting out' the program will be displayed.

The interactive elements of this approach allows for the easy, intuitive formation of solutions without needing technical knowledge of any particular programming language. If the solution is incorrect, the visualization will still attempt to run so the student can see the outcome and understand where they went wrong.

6.2 Visual Simulation

Upon the completion of their solution, a visualization of the execution of the input will be displayed even if their solution is incorrect. This system is expected to help students visualize the effects and flow of their recursive solution. The input form will need to be accurate and accept varying solutions. It will also need to determine minor syntax errors in the text input case.

6.3 Stack Visualization

A visualization of the call stack of the users' solution will be displayed. The visual will show how the call stack grows and shrinks during the execution of their solution as well as how values are passed between them. This outcome is to give the user a good understanding of how the stack operates. The visualization should accurately represent how the user coded their solution even if there are logic errors just as a regular IDE would.

7 PROJECT PLAN

7.1 Required Resources

All three game approaches will be developed in Unity. For the textual IDE, we will require an open source text editor that is implemented alongside the visual environment developed in Unity. Additionally, standard assets provided by the Unity store will be utilized. Unity's Web player will be used to create a browser application version for the tools.

7.2 Timeline

Week 1 (12 May – 19 May)
Finish Proposal

Week 2 (21 May – 28 May)
Finish Proposal Presentation

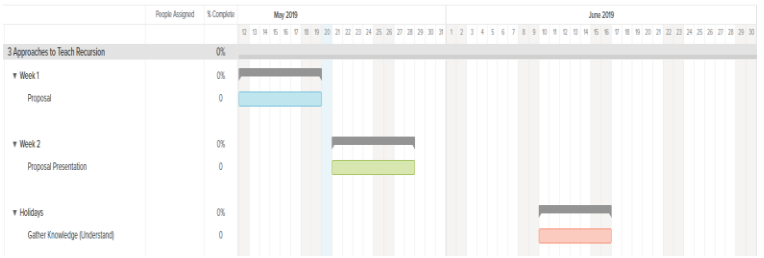
Holidays (10 June – 16 June)
Technology Frameworks Feasibility and Gather Knowledge (Understand)

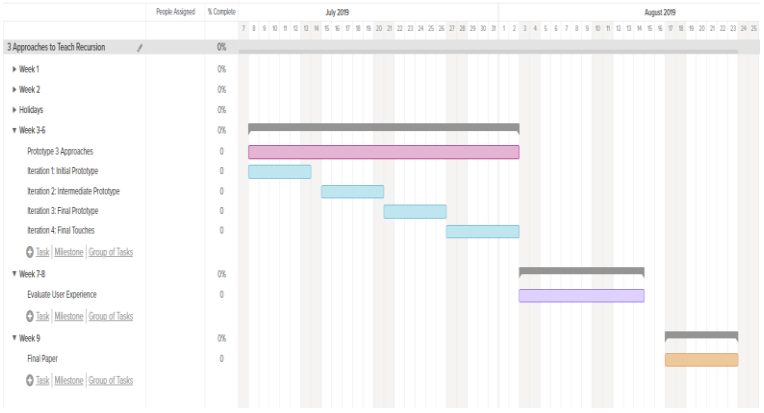
Week 3-6 (8 July – 2 August)
Prototype 3 approaches

- **8 July - 13 July**
  Iteration 1: Initial Prototype
- **15 July - 20 July**
  Iteration 2: Intermediate Prototype
- **21 July - 26 July**
  Iteration 3: Final Prototype
- **27 July - 2 August**
  Iteration 4: Final Touches

Week 7-8 (3 August – 14 August)
Evaluate User Experience

Week 9 (17 August – 23 August)
Final Paper





7.3 Deliverables and Milestones

Date	Description
Project Proposal Due	20/05
Project Presentation	27/05-29/05
Initial Software Feasibility Demo	15/07-19/07
Final Complete Draft of Paper	16/08
Project Paper Final Submission	26/08
Project Code Final Submission	2/09
Final Project Demonstration	2/09-16/09
Poster Due	23/09
Web Page	30/09
Reflection Paper	7/10
Open Afternoon/Evening	15/10

7.4 Work Allocation

Shakeel Mohamed	Responsible for the development of the stack visualization.
Moegamet Ra-eez Stenekamp	Responsible for the development of the visual coding approach.
Heng Jia (Tony) Guo	Responsible for the development of the visual simulation.

7.5 Risk Matrix

See Appendix A.

8 REFERENCES

1. McCracken, D.D. Ruminations on Computer Science Curricula. Communications of the ACM. 30, 1: (January 1987), 3-5.
2. Eagle, M., & Barnes, T. (2009). Experimental evaluation of an educational game for improved learning in introductory computing. ACM SIGCSE Bulletin, 2009, 321-325.
3. Tessler, J., Beth, B., & Lin, C. (2013). Using cargo-bot to provide contextualized learning of recursion. Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research - ICER '13. doi:10.1145/2493394.2493411
4. Chaffin, A., Doran, K., Hicks, D., & Barnes, T. (2009). Experimental evaluation of teaching recursion in a video game. Proceedings of the 2009 ACM SIGGRAPH Symposium on Video Games - Sandbox '09. doi:10.1145/1581073.1581086
5. 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
6. 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
7. Milne, I., & Rowe, G. (2002). *Education and Information Technologies, 7(1), 55–66*. doi:10.1023/a:1015362608943
8. Wu, C.-C., Dale, N. B., & Bethel, L. J. (1998). Conceptual models and cognitive learning styles in teaching recursion. Proceedings of the Twenty-Ninth SIGCSE Technical Symposium on Computer Science Education - SIGCSE '98. doi:10.1145/273133.274315
9. Dann, W., Cooper, S., & Pausch, R. (2001). Using visualization to teach novices recursion. ACM SIGCSE Bulletin, 33(3), 109–112. doi:10.1145/507758.377507
10. Kazimoglu, C., Kiernan, M., Bacon, L., & Mackinnon, L. (2012). A Serious Game for Developing Computational Thinking and Learning Introductory Computer Programming. Procedia - Social and Behavioral Sciences, 47, 1991–1999. doi:10.1016/j.sbspro.2012.06.938
11. Elenbogen, B. S., & O'Kennon, M. R. (1988). Teaching recursion using fractals in Prolog. Proceedings of the Nineteenth SIGCSE Technical Symposium on Computer Science Education - SIGCSE '88. doi:10.1145/52964.53029
12. Leroux, H., Réquilé-Romanczuk, A., & Mingins, C. (2003, June). JACOT: a tool to dynamically visualise the execution of concurrent Java programs. In Proceedings of the 2nd international conference on Principles and practice of programming in Java (pp. 201-206). Computer Science Press, Inc..
13. IJsselsteijn, W. A., De Kort, Y. A. W., & Poels, K. (2013). The game experience questionnaire. Eindhoven: Technische Universiteit Eindhoven.

Appendix A

	Risk Description	Consequence	Probability (1-10)	Impact (1-10)	Factor	Mitigation Strategies
1	Bugs that don't appear at run time.	This may affect the users experience using the system, which could negatively affect the results of their evaluation.	4	6	24	Thorough testing must be conducted before releasing the system to be evaluated. This will find any bugs and allow us to fix them before any user can encounter them.
2	Development of a system takes longer than planned.	This may result in the system not being ready in time to be evaluated, resulting in that system not being able to produce results for this project.	6	8	48	Plan extra time in addition to the estimated time to complete the system. This will ensure that if there are any delays then there will be sufficient time to still complete the system. If there is still not enough time, then the core functions of the system will have to be identified and focused on completing in the remaining team, leaving out any non-essential functions and features.
3	Scheduled deadlines not met.	An incomplete, low quality system will be created.	3	7	21	Realistic and attainable goals and deliverables must be set not too far apart. This will allow for a Gantt chart to be created which, when used alongside the deliverables, will ensure that the creation of the system does not fall behind schedule. Frequent scheduled meetings will help ensure that the system is following the planned schedule.
4	Gold plating a system (adding too many out of scope and unnecessary features)	This may slow down development time and may take away from the core functions of the system that the user will need to help their understanding of recursion.	2	3	6	Review scope frequently to ensure that the system is within scope and to correct when development begins to stray from the scope.
5	If group members fall behind on their specific portion of work.	This will result in the work not being complete and thus the different approaches won't be ready in time to have their user experiences tested.	3	8	24	Regular group meetings to discuss progress. Members help each other and distribute workload evenly.
6	Technologies chosen to use are not adequate to develop the systems. Technology too difficult to use.	A new technology will have to be chosen. This would result in a lot of wasted time and could put the project behind schedule.	3	9	27	Thorough research done before choosing technology to use for developing the systems. Ensuring the technology supports the system to be built