

aMazeChallenge – An interactive game for learning to code

Nicos Kasenides

30 April 2018

**A project report submitted in partial fulfilment for the degree of
Bachelor of Science in Computing**

**School of Physical Sciences and Computing
University of Central Lancashire**

Abstract

In a rapidly growing digital realm, the supply of programmers and software engineers determines the speed with which we move our technology forward. Unfortunately, the supply today is inadequate to fill the demands for such positions. A great challenge lies ahead to attract new students to computing and to create a technologically literate world, balancing the demand and supply of these vacancies. This study attempts to reveal how educational programming games can help create a platform with which young students can learn programming in an effective entertaining way. Features of such games are examined from past research to design a learner-based system. Different methodologies have been studied and the advantages or disadvantages of each of them have been outlined. Moreover, the predisposition of students towards programming is considered as a factor affecting the opinions of the general population in regard to this field and judgements have been made using existing literature. An Android maze-solving game was created to carry out this idea and test these hypotheses. A group of students not particularly interested in computing were given a small lecture on basic programming principles and were then asked to write their own code to solve a maze challenge using a visual programming language called Blockly. A questionnaire was answered by these students before and after taking part in this game, which resulted in mixed outcomes. These outcomes showed some expected and unexpected results. For example, compared to other studies this group did not have a high interest in programming prior to playing the game. Comparisons were made with a study conducted last year during Code Cyprus 2017. These comparisons show that both voluntarily enrolled and non-voluntarily enrolled students performed poorly in terms of programming skills and that their perception of programming was the only difference. More importantly, the present study suggests that the majority of non-interested students would still consider studying programming despite the difficulties faced. This evaluation confirms various other studies that suggest videogames can be used successfully as an alternative method to teach programming. Moreover, it also suggests that a group of not particularly interested students can be convinced to consider studying a programming related course and that aMazeChallenge has possibly helped align them with this opinion.

Attestation

I understand the nature of plagiarism, and I am aware of the University's policy on this.

I certify that this document reports original work by me during my University project.

Signature

Date

Acknowledgements

First and foremost, I would like to thank my supervisor Dr Nearchos Paspallis for his decision to involve me in this project. Dr Paspallis organized important events regarding this project and has been an ideal mentor not only during its duration but also the entire course by being approachable, helpful, kind, honest and sharing a tremendous amount of knowledge.

I would also like to thank the University of Central Lancashire Cyprus for providing the necessary tools to carry out this project as well as the lecture hall in which a major part of this project's evaluation occurred.

The participation of the students from Livadia High School was instrumental in extracting important insights related to the game's performance and effectiveness. I am thankful for their participation in this study.

Finally, I would like to thank my family and my classmate and girlfriend Panayiota, who have supported me during the project.

Table of Contents

Abstract.....	i
Attestation.....	ii
Acknowledgements	iii
Table of Contents	iv
List of Figures.....	viii
List of Tables	x
List of Listings.....	xi
1 Introduction.....	1
1.1 Background and Context.....	1
1.2 Scope and Objectives	2
1.2.1 Scope.....	2
1.2.2 Objectives	3
1.3 Achievements	6
1.4 Overview of Report	6
2 Literature Review	7
2.1 Introduction	7
2.2 Conventional educational methods vs Games.....	7
2.3 Useful features of educational games	11
2.4 Methods of evaluating the efficiency of educational games	13
2.5 Summary.....	15
2.6 Hypothesis	16
3 Project Planning.....	17
3.1 Introduction	17
3.2 Methodology	17
3.3 Requirements.....	19
3.4 Potential Solutions	20
3.4.1 Programming Language.....	20
3.4.2 Platforms and devices	22
3.4.3 Gameplay, Graphics and Audio.....	23
3.4.4 Online hosting	23
3.5 Costs.....	24
3.6 Tools and Techniques	24
3.7 Legal and Ethical Issues	26
3.7.1 Electronic personal data	26
3.7.2 Questionnaire personal data.....	26

3.7.3 Other issues	26
3.8 Summary.....	26
4 Design.....	27
4.1 Introduction	27
4.2 System Characteristics.....	27
4.3 System Architecture	27
4.3.1 Application level	27
4.3.2 System level.....	28
4.4 System Design	29
4.4.1 Application flow	29
4.4.2 Game rules and mechanics.....	30
4.4.3 State representation	32
4.4.4 Data Structures and Algorithms.....	34
4.4.5 Web Services / API	38
4.4.6 Datastore.....	38
4.4.7 File system	39
4.5 User Interface Design.....	39
4.6 Summary.....	39
5 Implementation	40
5.1 Introduction	40
5.2 Programming standards and conventions	40
5.2.1 Language Conventions.....	40
5.2.2 Readability, reusability, maintainability and expandability	40
5.3 Client-side	41
5.3.1 Maze and player visualization.....	41
5.3.2 Interacting with Blockly code	41
5.3.3 Maze Designer and types of mazes.....	43
5.3.4 Training Mode and maze controls.....	43
5.3.5 Gamification of maze-solving.....	44
5.3.6 Personalization.....	44
5.3.7 Learning	44
5.3.8 Online Mode using a Web API.....	44
5.3.9 Main menu and user preferences	45
5.3.10 Questionnaires	45
5.3.11 User interface.....	45
5.4 Server-side.....	46
5.5 Summary.....	46

6 Test Strategy	47
6.1 Introduction	47
6.2 Unit Testing.....	47
6.3 Integration testing.....	48
6.4 System testing.....	48
6.5 Acceptance testing	49
6.6 Summary.....	49
7 Evaluation, Conclusions and Future Work	50
7.1 Introduction	50
7.2 Project Objectives	50
7.3 Evaluation	50
7.3.1 Player feedback.....	50
7.3.2 Objective attainment.....	53
7.3.3 Difficulties faced	53
7.3.4 Usefulness of solution and comparison with similar work	53
7.4 Applicability of Findings to the Commercial World	54
7.5 Conclusions	54
7.6 Future Work	55
7.7 Concluding Reflections.....	55
References	56
8 Appendices	59
8.1 Appendix 1	59
8.2 Appendix 2	59
8.3 Appendix 3	60
8.4 Appendix 4	62
8.5 Appendix 5	63
8.6 Appendix 6	64
8.7 Appendix 7	65
8.8 Appendix 8	66
8.9 Appendix 9	67
8.10 Appendix 10	68
8.11 Appendix 11	69
8.12 Appendix 12	72
8.13 Appendix 13	74
8.14 Appendix 14	75
8.15 Appendix 15	76
8.16 Appendix 16	77

8.17 Appendix 17	79
8.18 Appendix 18	80
8.19 Appendix 19	82
8.20 Appendix 20	83
8.21 Appendix 21	84
8.22 Appendix 22	85
8.23 Appendix 23	86
8.24 Appendix 24	87
8.25 Appendix 25	88
8.26 Appendix 26	89

List of Figures

Figure 1 - A bar chart comparing available jobs and graduates in STEM fields in the US	1
Figure 2 - Factors that influence participant's intrinsic motivation according to Long (2007).....	10
Figure 3 - The performance of the two groups after three experiments (more is better).....	14
Figure 4 - Stages of the project.....	17
Figure 5 - Diagram comparing the impact of traditional and agile approaches on a project's quality.....	18
Figure 6 - Client-Server Architecture.....	25
Figure 7 - The MVC architecture.....	28
Figure 8 - The system architecture based on the client-server model	28
Figure 9 - A flowchart representing the game's flow from the menu to different views.	29
Figure 10 - A diagram showing how players' turns are executed.	31
Figure 11 - A state transition diagram representing the game state.	32
Figure 12 - A state transition diagram showing the state of picked objects	33
Figure 13 - A state transition diagram depicting bomb states and the transitions between them.....	34
Figure 14 - Fragment of a class diagram and CRC card showing how color could be represented in the game.....	35
Figure 15 - Levels of testing	47
Figure 16 - Code designer and static checker state transition diagram	59
Figure 17 - Main Menu design	60
Figure 18 - Learning activity design	60
Figure 19 - Personalization activity design.....	60
Figure 20 - Tutorial Activity design.....	60
Figure 21 - Game Activity design.....	61
Figure 22 - Maze designer design	61
Figure 23 - Blockly code library.....	63
Figure 24 - An example implementation of the left wall follower algorithm in Blockly	67
Figure 25 - A set of screenshots demonstrating the generation of different types of mazes.....	68
Figure 26 - A screenshot showing the generation and drawing of pickable objects.....	70

Figure 27 - Screenshots showing the implemented personalization activities and ViewPagers.	73
Figure 28 - Screenshots showing several help/learning pages creating using HTML/CSS.	74
Figure 29 - A screenshot showing the implemented questionnaire activity.....	75
Figure 30 - Screenshots showing the implementation of the user interface.....	76
Figure 31 - A pie chart showing the distribution of gender during the high school visit	82
Figure 32 - A pie chart showing the distribution of gender during Code Cyprus 2017	82
Figure 33 - A bar chart showing self-rating of students related to mathematics and programming during both events.	83
Figure 34 - A bar chart showing the differences between opinions on programming before and after the two events	84
Figure 35 - A bar chart showing reasons for interest in programming by students. .	85
Figure 36 - A bar chart showing the difference in programming skills before and after the game.	86
Figure 37 - A bar chart showing the student's perceived rating on their own programming skills after the game.....	86
Figure 38 - A bar chart showing the most understood concepts by students.....	87
Figure 39 - A bar chart showing the least understood concepts by students.....	87
Figure 40 - Chances of following a programming career.....	88
Figure 41 - A pie chart showing the most enjoyed features of aMazeChallenge	89

List of Tables

Table 1 - A table of MoSCoW priority arrangement for the project's objectives.	5
Table 2 - A table showing average times of compilation across different devices. ..	48
Table 3 - A table showing objective attainment for aMazeChallenge.	81

List of Listings

Listing 1 - onDraw() in GameView.java.....	62
Listing 2 - Static checkers code	64
Listing 3 - processCode() in InterpreterMazeSolver.java.....	65
Listing 4 - Code interpreting the player's code.	66
Listing 5 – handlePickableState() in RuntimeController.java	69
Listing 6 - applyPlayerMove() in RuntimeController.java.....	69
Listing 7 – Code showing the use of preferences to save personalization data.....	72
Listing 8 - Code demonstrating the use of ViewPagers to select icons and colors. .	73
Listing 9 - Code showing how enumerating classes are used to represent types of responses for the questionnaire.....	75
Listing 10 - Code showing how the server queues an instruction for the game's engine to run.	77
Listing 11 - Code showing how the server handles players by organizing them into queues for games.....	78
Listing 12 - Code showing how a JUnit test is used to test the questionnaire submission.	79

1 Introduction

1.1 Background and Context

As a person who embraced Computer Science and programming from a very young age, I always believed that I was not an outlying case but rather a case for which the necessary motivation and tools had been provided. Unfortunately, while not outlying, my personal case belongs to a minority. Even in today's digital age, in which we all bear witness to the amount of prosperity in which our understanding of computing has resulted, the choice to earn a degree in this field is rather infrequent.

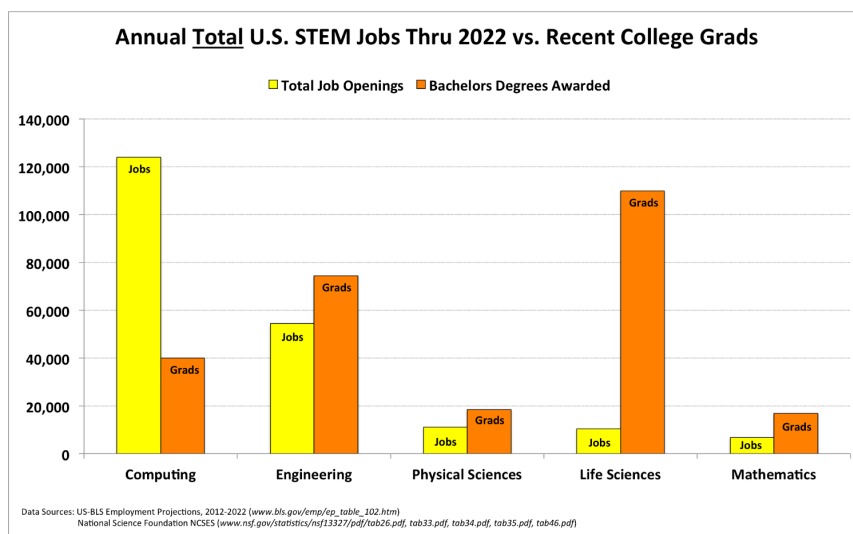


Figure 1 - A bar chart comparing available jobs and graduates in STEM fields in the US

A study of the National Science Foundation in the United States (Figure 1), reveals in this case the outstanding difference between the expected demand for computing jobs compared to the number of computing graduates in 2022. In fact, according to this study, all STEM fields have a higher supply of graduates than the demand for jobs with the single exception of computing. Also important is to note that the number of jobs in computing expected to be available in 2022 is at least double of any other STEM field. There is already a shortage of software engineers, programmers, developers etc in today's market that needs to be mitigated and according to (Paspallis, et al., 2018), *"one of the main factors contributing to this shortage is lack of motivation of young people to learn programming"*.

There are certain prejudices that cast a shadow on the popularity of computing and programming. First and foremost, people (particularly high school students) tend to find programming to be a daunting task. Modern media and culture have portrayed programmers as people with above average intelligence who are able to think and type fast and seemingly solve unsolvable problems. While it can be argued that anyone studying computing is likely

to improve their problem-solving skills, the case of programming being a difficult practice is certainly not true. The argument of highlighting the importance of computing needs to be made at an age when students are aligning themselves with a particular study. Unfortunately, at this age students generally have a short attention span. For example, a student may get excited about robotics or videogames but their interest will quickly fade away once they get into a classroom where programming has nothing to do with what they were previously interested in. This begs the question of how programming can be made fun and relate to what students are initially interested about. After all, an educator will almost certainly have difficulties convincing students to study computing so that they develop an algorithm that is 10% faster, but their chances of getting the necessary attention increase if they relate programming to an entertaining and creative task such as creating a smart robot, playing an immersive videogame or any other such tasks.

1.2 Scope and Objectives

1.2.1 Scope

aMazeChallenge falls in the category of videogames as it is an interactive multiplayer game for mobile devices. It features different maze scenarios from which the players need to escape by moving their avatar in a virtual maze arena from the starting to the exit point. However, in this game the avatar is not controlled by any user interaction such as touching or tilting the phone but by writing code, which causes the avatar to move. This code is written in Blockly¹, a visual block language provided by Google that is easy to learn and understand. Consequently, players need to write their own code to escape the mazes and thus learn, understand and acquire knowledge on how to write basic code. Despite being available to play by anyone and providing challenges even to experienced programmers, aMazeChallenge is mainly destined for high school students who have little or no experience in programming. The game is accompanied by tutorial/help pages which teach players how the game works and the basic concepts of programming. Beginners can start learning programming within the game itself, expand their knowledge as they make progress and practice what they learn by playing. Therefore, such a game needs to be simple so that beginners can understand how to play. The timeframe of this project is a major limiting factor which affects any decisions throughout. However, the requirements of the project need to be achieved and this limitation should not have any effect on either meeting the set objectives or the necessary quality. Several implementation requirements of this project are to:

1. Be hosted on a cloud-based service.

¹ A library for building visual programming editors: <https://developers.google.com/blockly/>

2. Visualize the game progress and player standings in real time.
3. Allow players to register to the game, write and test their code.
4. Realize appropriate game elements such as graphics and sound.

aMazeChallenge will predominantly be used during educational events such as Code Cyprus and other fairs, but it can be downloaded from the Google Play Store² by anyone and is free to play.

1.2.2 Objectives

aMazeChallenge aims to challenge known prejudices about programming by showing that it can be a fun, creative and enjoyable task. By allowing players to view how the code they have written behaves in the game using the movement of their avatar as a reference, beginners can learn how to:

1. Read and understand pre-written code.

This is usually the first experience of any beginner into programming. It is almost impossible for anyone to write code in any language without first understanding how its syntax and semantics work. The objective in this case is to transition from real life actions (such as move forward, turn clockwise, look left etc.) which are naturally more understandable to in-game commands that represent them.

2. Improve or fix pre-written code.

The players are asked to incrementally improve their code during the training process. The difficulty of the mazes in this mode increases with each level. Players are asked to make modifications to their own code to make it work in the new, more difficult maze. In addition, the already existing code samples need to be edited so that they work in various kinds of mazes. This is a skill that is necessary for any future programmer.

3. Write their own code from scratch.

After learning to read, understand and fix existing code, any programmer needs to be able to write code from scratch without having to reference existing code (with the exception of any possible built-in functions). The code editor allows the players to write their own code from scratch without limitations.

² Download from Google Play Store:

<https://play.google.com/store/apps/details?id=org.inspirecenter.amazechallenge&hl=en>

4. Improve code in terms of efficiency and clarity.

Although this is not a main requirement for a beginner, it is an important aspect when learning to program. It is preferable that the qualities of code efficiency and clarity are highlighted from the very start of the learning process so that they are correctly practiced, used and adopted.

By practicing the aforementioned programming skills, players will most importantly develop critical thinking and problem-solving skills that are applicable not only in programming and computing but in all aspects of life. A missing link in this case is making students aware of their “subconscious” problem-solving skills (skills that they might have used to solve problems in real life or in other situations without really knowing it) and enabling them to use these skills in a productive manner. aMazeChallenge is a competitive game and tries to leverage a person’s competitive motivation to learn programming. Hopefully, this motivation to compete with others is also translated into motivation to learn programming and will help increase student retainment.

Beyond the attempt to teach basic programming, this project is also focused on evaluating how this approach performs compared to traditional approaches used in education. Different teaching techniques used in programming will be studied from existing literature to provide insights on how other people around the world have attempted to solve this problem. Comparisons will be made between these techniques which will be compared explicitly with the method tried in this project. Such an evaluation will take into account different factors that affect learning, specifically in younger generations but also how teaching computer programming is taught differently from other subjects.

Perhaps the most important objective of this project is to make people interested in computer programming and let them discover that it is an entertaining task they already know how to perform subconsciously. Such a task is of tremendous difficulty, especially if the mentality of the general public is taken into account regarding the field of computer science. The most difficult challenge is in fact how to reach out to people who have no perception of what programming is or even worse “*have mistakenly developed a negative picture for it*” (Paspallis, et al., 2018). Obtaining insights on what the general population thinks and possibly engaging them in programming lies at the core of this project.

The following is a table of the implementation objectives for aMazeChallenge arranged using the MoSCoW method (Clegg & Barker, 1994):

Objective		Priority
1	Implemented on an appropriate cloud-based platform such as AppEngine.	MUST HAVE
2	Provides a view which visualizes the game progress and players' standing in real time, implemented on an appropriate platform.	MUST HAVE
3	Provides a mobile view which enables the players to register to the game, define and test their code, and view their progress, on an appropriate platform for mobile devices such as the Web or Android	MUST HAVE
4	Realizes appropriate game elements, including graphics and sounds which enhance the game experience.	MUST HAVE
5	Includes interactive elements such as avatars, icons with colors that represent the player and can be changed in a personalization screen.	MUST HAVE
6	Incorporates a language that defines how the player will move and interact with objects inside the maze.	MUST HAVE
7	Allows a player to learn about the game, its rules and how it is played using a training scenario and help pages.	SHOULD HAVE
8	Utilizes a language that is highly usable for mobile devices and appropriate for beginners.	SHOULD HAVE
9	Uses a built-in mechanism that checks the player's code for obvious errors.	SHOULD HAVE
10	Include basic preset algorithms that can be used by players to compete against or changed to write their own code.	SHOULD HAVE
11	Include features that promote the game experience such as different objects that affect the player's health or points (e.g time bombs, traps, fruits and more)	SHOULD HAVE
12	Allows the player to control the sound and vibration by toggling them on or off.	COULD HAVE
13	Features multiple languages and allows the player to change between them.	COULD HAVE
14	Features a maze designer that allows players to design and play their own mazes.	WOULD HAVE
15	Includes video tutorials that explain how to play and write code.	WOULD HAVE
16	Uses 3D graphics and more advanced game elements such as animations to enhance the game experience	WOULD HAVE

Table 1 - A table of MoSCoW priority arrangement for the project's objectives.

1.3 Achievements

Nearly all of the implementation goals of this project have been completed. The game that was created performs well and was used to give a good example to a group of students on what programming is and how it works. During the evaluation the data gathered has helped understand the opinion of these students on programming and if they would be willing to study a related course. Despite not showing whether or not actual programming skills increase after the game, the data gathered is important in understanding how to shape the students' opinions to develop even better tools or games which can be increasingly more student friendly and have a better chance of guarantying success.

1.4 Overview of Report

In this report we analyse the current methods used to teach computing and programming, using several sources and comparisons. A plan of execution follows outlining the requirements of the project, different methods that can be followed, steps taken to design and develop aMazeChallenge as well as how to evaluate the method being used. A detailed design of the game discusses how the system's components are connected, how the game can be used by players along with its rules and mechanics. The implementation section overviews how the entire system was implemented and the techniques that have been used throughout. Following that, we discuss the strategy adopted to test the game and make sure it performs satisfactorily. An evaluation takes place in the following chapter, discussing how results were gathered, what they reveal and how that can be interpreted in the larger context of learning programming. Lastly, we list the conclusions and argue on their implications in academic and commercial environments.

2 Literature Review

2.1 Introduction

In order to fully comprehend the value and efficiency of teaching programming through games, it is important to have information on the following:

1. The current methods that have been used in the past or are currently being used today to teach programming through games.
2. An understanding of any advantages, disadvantages, risks and rewards that these approaches offer.
3. Details on how these methods have been implemented and how they have performed.

2.2 Conventional educational methods vs Games

The idea of using videogames as a means to educate students is not a new concept. It causes some to “*leap for joy*” and some to “*ask questions about this learning medium*” (Klopfer, et al., 2009). Many of those who are interested in creating such educational games do not know if the final product will be worth the effort. Even after making the decision to develop an educational game, it is hard to decide where to start, as there are so many ways to implement such games. The authors highlight that “*[Some] group[s] see the skills students develop playing games as essential to a 21st century education, and conversely see little progress happening in schools still shackled to a 19th century factory model*” (Klopfer, et al., 2009). On the other hand, there are some concerns that need to be addressed in order to integrate games into school environments. Such educational games need to cover the content that is originally taught using standard teaching methods. Secondly, a game needs to be competent enough to attract student’s attention so that the learning process can be achieved and maintained. Contrary to normal classes, games can be played “*in very short bursts of class time*” and seem simpler to students compared to workbooks, lectures and other conventional methods. In addition, (Klopfer, et al., 2009) also say that games allow more freedom to students:

1. Freedom to fail;
2. Freedom to experiment;
3. Freedom to fashion identities;
4. Freedom of effort;
5. Freedom of interpretation.

Additionally, games can be a fun way to learn and have the advantage of appealing to a mass market which may help spread the popularity of what is being taught – in this case programming.

The ability to program a computer or system requires a high amount of creativity and sometimes imagination. However, according to (Vinicius & James, 2016) “*the number of pedagogical methods addressing creative aspects of programming is low in computer science research*”. Beginners in programming often show a lack of skills related to creativity, which severely impacts the ability to solve problems. The authors argue that games can be “*strong allies*” in addressing this issue because the students’ mental and intellectual capacity is increased through retention and motivation. Students can participate in “*especially conducive situations*” where interactions are frequent and students take part as a whole in a creative class. A study was conducted involving thirty students in the first year of secondary school using two different methods. In the first method, the students were presented with nine programming patterns in a computer lab. The teacher asked the students to solve these exercises using C++. The creative aspect of this experiment comes from the fact that in the second method, students also had to implement small games using concrete materials in a sports field. These two contrasting approaches attempted to evaluate the use of programming patterns and how efficiently each method helped students learn them. A questionnaire answered by students revealed that all students preferred “*an alteration in learning environments*” (referring to the concrete materials game approach) and said that it could be profitable for learning these programming patterns. The students’ answers also revealed that they were “*not motivated to study programming in a traditional way*”. Another questionnaire conducted by this research related to user experience discovered that all students liked the fact that they had participated in a game and nobody said that “*the activity was indifferent, hated or disliked*”. More importantly for the creativity aspect, only one student replied that they were indifferent during the game’s creation process and almost all students (97%) said that they felt either happy or excited during the creation of games. Furthermore, in terms of how games benefited to learning programming, no students replied that “*the activity did not add anything or added little*” to their experience, with most answers ranging from “*something to a lot*”. Another advantage of using games to enhance creativity this research reveals is that interaction between students in the same group was much higher than with those of other groups. According to (Vinicius & James, 2016), this means that “*students [...] were more motivated to interact within the game created by them than in the games created by other[s]*”. Ultimately, using this creativity-based approach the teachers were able to increase the average performance of the students by 17.42% - a very significant amount - thus confirming that creativity plays an equally significant role in student

interest, learning and retainment. This research was conducted among high school students who were beginners in programming and therefore perfectly matches the setting for aMazeChallenge. This makes it a valuable source in identifying how differences between conventional methods of teaching and learning through games affect student performance and satisfaction, even though the game devised was not a computer game.

Another research conducted by (Harris & Jovanovic, 2010) shows how changing an entire university programming course could potentially affect student motivation, retainment and skills. The course at Georgia Southern University was originally designed to “*help students without any programming experience*”. In this approach students used Visual Basic and the Visual Studio IDE³ to create a variety of different games throughout the course, applying their programming knowledge as a problem-solving tool. The objective of the new approach was to motivate students to learn by creating ever more difficult games, starting with striker and dice and moving on to hangman, blackjack, Tetris and space invaders. Despite the effort to create such a course, the report does not comment on any of its effects, results or conclusions. However, it is useful because it shows the lengths at which high-level institutions are willing to go to integrate games into the learning environment. Moreover, it uses a different approach to using games compared to others. Students are not asked to play a game but are asked to create one. Such a method might have some peculiarities in terms of advantages and disadvantages. However, because this is a university course in which the objective is to learn the basics of programming as quickly as possible, this approach could be useful in making the process more entertaining and perhaps help students keep their interest in the course. The differences with aMazeChallenge are obvious in this case since the two projects deal with different ages of students and use games as a teaching tool in a very different way. Despite that, in both cases the students are beginners in programming and therefore it is acceptable to say that this is an alternative method of utilizing games in learning to program.

Confirming ideas from the above sources, (Long, 2007) says that “*one of the primary tasks educators face is to motivate learners*”. Long attests that games can:

1. Enhance the self-esteem of learners;
2. Reduce training time and instructor load;
3. Provide more opportunities for practice;
4. Enhance knowledge acquisition.

This study focuses on a game called IBM Robocode, which has its own diverse community of players and attempts to reveal how this game influenced the participant's learning

³ Integrated Development Environment:
https://en.wikipedia.org/wiki/Integrated_development_environment

outcome, programming skills, if the game was enjoyable and if yes, which activities in the game were enjoyed most. Robocode is a game that teaches participants how to use Java to create and program their robot in order to participate in games and battle *“against other robots in a battle arena”*. A survey was conducted with eighty-three valid responses in which 80% of the participants said they felt that their programming skills had increased after playing Robocode. Because of a range in demographics, the survey also revealed that the game was not only enjoyed by the younger (and less experienced) audience but also by the older (and often more experienced) audience. The motivation stemming from the participants was primarily because of intrinsic motivators such as *“to have fun”* or *“to learn new programming skills”*. On the contrary, extrinsic motivators such as *“to win the game”* or *“win a prize”* were proven to be less significant.

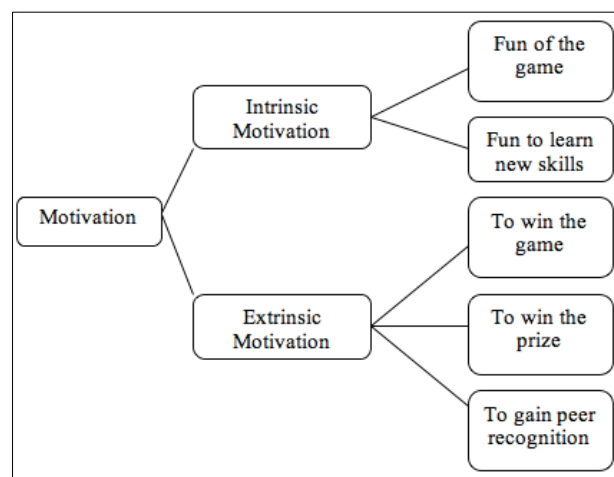


Figure 2 - Factors that influence participant's intrinsic motivation according to Long (2007)

In addition, the most frequently chosen factor that made Robocode fun was *“[the ability] to solve problems on my own”*. According to the author Robocode is a *“real-world programming game with real participants, increasing the validity of [their] research”*. The use of real participants in such experiments is indeed a factor that benefits the validity of the results. In addition to that, this experiment was conducted among a much more diverse community (in contrast with aMazeChallenge) which also included experienced programmers. This was useful in determining the traits that made an experienced programmer, while still maintaining fairness by splitting the contestants into different groups based on their programming level (Beginner, Intermediate and Advanced). Furthermore, *“participants all worked on the same software on the same platform”*, which is similar to the proposed solution for aMazeChallenge. This helps ensure that the development tasks *“[are] similar for all participants”* and thus reduces the extent *“to which different programming languages [...] could act as confounding variables”*. The depth in which this research operates along with

the reasons stated above make it a very useful provider of information. The report by Long reveals that games can be used by instructors to “*provide students with substantial opportunities for creativity*” and says that the Robocode results are likely to “*generalize well [...] [for] higher-level courses or even graduate courses*”.

2.3 Useful features of educational games

Videogames have spawned a multi-billion-dollar industry in the twenty-first century and are not expected to go away any time soon. The rise of portable devices such as tablets and smartphones as well as social media has helped the world of videogames go viral around the world. Games have a distinct way of immersing people into virtual worlds in which they can behave in almost absolute freedom. Ironically, this is a shared aspect between games and programming, however no comparison can be made in terms of their popularity. In their report, (Seng & Wan, 2015) enumerate the core “*constructs*” of educational videogames:

1. Playability;
2. Multimedia adequacy;
3. Ease of use;
4. Enjoyment;
5. Immersion.

As the authors say, “*The 21st century sees a new group of younger [...] generations who grow up with and are exposed to different devices*”. Because of that, the authors argue that “*[students] need to be educated in a similar manner*”. Overall results of this research showed that in a game related to programming concepts, “*78.9% of students agreed or strongly agreed that they were able to become quickly familiarized with the game*”.

(Long, 2007) says that educational games “*should be kept simple, since the learners’ thresholds of interest and concentration may be low*”. Similarly, the instructors of the games should also be kept simple so that the game does not cause frustration among its players and so that learning the rules of the game is neither difficult or time consuming. Long also says that instructors should modularize game tasks and provide rewards for each subtask, leading to constant gratification for the learner. This in turn could lead to “*increase[d] sense of autonomy and competence*”. The length of modules should vary between short and large modules so that the frequency of satisfactory outcomes is balanced with retained involvement. Lastly, Long suggests that such a game should provide “*different levels of challenges*” to make sure that it is easy to start but challenging enough to preserve the player’s interest in the long run.

Another similar case is Game2Learn (Barnes, et al., 2007), a Research Lab that specializes in developing games to *“improve recruiting and retention in computer science through [...] game-based learning environments”*. The authors have developed several games, some of which they have used to conduct experiments related to student learning especially in Computer Science. Consistently with the previous source, Game2Learn leverages games to counter negative experiences of students related to the field and attempts to provide evidence that an educational game can be fun and more engaging to students than conventional methods of education while still maintaining the same syllabus. Game2Learn proposes *“apply[ing] games in the computer science curriculum through playing an MMORPG”* (Massive Multitplayer Online Role-Playing Game) and coding in *“iconic programming languages”*. Such a visual and immersive game is expected to attract students into playing and learning simultaneously. Furthermore, the way the game is played and how the students are supposed to interact with it and write code has been carefully chosen through series of prototyping processes. This suggests that the game has been optimized to suit the requirements of its educational objectives as much as possible. However, an obvious difference of these experiments compared to aMazeChallenge is that it utilizes teamwork during the game. Another major difference is that Game2Learn uses both 2D and 3D games. In these games, features such as music, storylines, game mechanics and visual effects are included to enhance the player’s experience. In fact, these games appear to be more entertainment rather than educational, something that may positively impact the learning capacity of the participants. In addition, players can directly see the effects of bad code: *“When the player makes a mistake, Arshes must fight a script bug, which asks the users various computer science questions in order to fight the bug”*. The game tasks grow progressively more complicated - parallel to what is suggested by (Long, 2007) - and involve reordering or correcting statements to achieve a goal and writing simple code statements. If incorrect answers are given an indicative reaction occurs in the game such as decreasing the player’s health. Evaluations of these games *“provide evidence that a game for learning programming can be fun, engaging and satisfying for students”*. However, this source also exposes some limitations of this approach, because *“some [students] were unsure if the game tasks would teach them enough”*.

A study made by (Leong, et al., 2011) talks about a self-paced learning system called “Paths”. This system uniquely utilizes social interactions and intensive feedback to the students to improve their learning experience. It includes a feed-based system with which students are able to discuss their assignments in a conversational manner. This allows them to reflect on their progress, during an interactive game. In fact, the Computer Science department at the University of Singapore converted a *“module [...] into 22 missions which*

were framed around a storyline in a Star Wars-based universe". This game also features game mechanics and in addition includes rewards for task completion. A main difference of this experiment relative to aMazeChallenge and the majority of sources is the fact that the participants of this experiment were university students. Nevertheless, the source suggests that a lot of these students were beginners in programming. The features of this game gamify the course and feature a scoreboard with which students are introduced to a competitive environment (similar to aMazeChallenge's proposed functionality). The system also rewards students for attending classes and is linked to Facebook: *"feeds are posted onto Facebook when students level up or unlock certain achievements"* which helps improve student engagement. During the evaluation of the game *"71% of the students found that [the] new approach made the course much more interesting compared to traditional courses, and the average submission times for assignments improved"*. Therefore, the results express that educational games can improve student interest, retention, performance and punctuality in terms of meeting deadlines. The experiment was conducted using several online forums and surveys. However, this source also suggests that limitations may exist to this approach, because *"some students feel more stressed as they feel they have a lot of deadlines to meet and a lot of assignments to submit, even though they have a similar amount of work"*. This warns against introducing an overwhelming number of tasks and game features. (Chan, 2014) also says that during her own study *"some students felt that there was too much chaos introduced during gameplay"* which appears to be consistent. This hints that there should be a balance in the game and that it shouldn't be too stressful or lead to chaos. Lastly, Chan points out the importance of using interactive tutorials to teach students how the game works, how they are supposed to write their code and most importantly, illustrate the different programming concepts.

2.4 Methods of evaluating the efficiency of educational games

(Eagle & Barnes, n.d.) have also developed an educational game called "Wu's Castle", a 2D game where university-level Computer Science students who were attending the introductory programming module were asked to implement changes to already existing programs in an interactive and visual way. This game provides immediate feedback and students are able to visualize code execution. It is worth pointing out that the code in this game is composed from visual elements and not text which generally makes it easier for a beginner to understand and write. A main difference with several other games and aMazeChallenge is that students were tasked with very specific changes related to just arrays and loops. This limits the scope of the game to just learning these concepts and not a

general idea of programming. During the evaluation of this game the participants of the course were split into two groups: a control group and a group that played the game. This random selection and the use of a control group allowed the researchers to examine how the results differed between the two. *“Results show[ed] that students who play [the] learning game first outperform those who write a program before playing the game. Students in the game-first group felt they spent less time on the assignments, and all students preferred the learning game over the program”*. Despite being related to a university course and its limited scope, this experiment suggests that features of a game *“can help prepare students to create deeper, more robust understanding of computing concepts while improving their perceptions of computing homework assignments”*. The results of the evaluation are shown by the graph below:

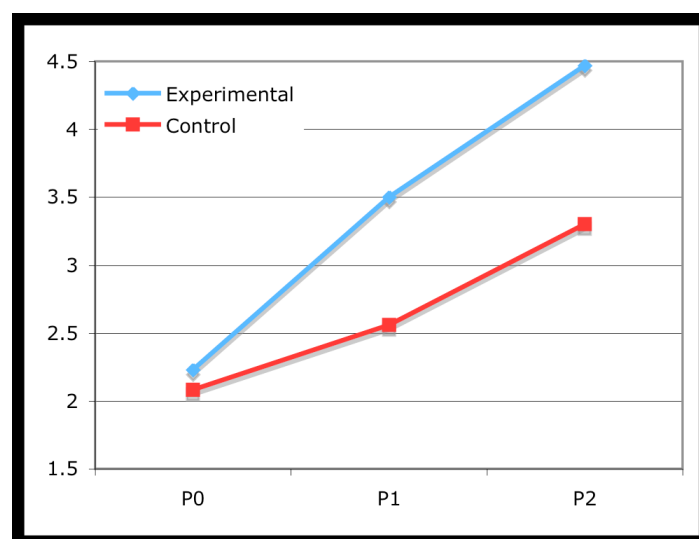


Figure 3 - The performance of the two groups after three experiments (more is better)

In her graduate thesis, (Chan, 2014) also uses a similar game called “Space Race” *“to encourage students to share their individual understandings of basic programming concepts”*. She uses *“the power of group discussion”* in a highly competitive game to engage students in a programming environment where they learn through *“self-exploration and experimentation”*. In the same way as most other sources, Chan utilizes a 2D game, but particularly focuses on participant experimentation to run her tests. In addition, she mentions that *“students were observed to learn more effectively in groups”* and uses teamwork in her game to take advantage of this fact. During the evaluation of this game, she attempts to highlight the importance of feedback to students, demonstrate how cooperation and social interaction can be used effectively and reveal that *“programming knowledge obtained in a video game can be used in a nongaming context”*. The main highlight of the results of this experiment is that *“At least 67% of the students stated that the game motivates them to*

review course material". While this is not a very high percentage, it is still a positive one. Chan deduces from her experiment's results that *"in some cases, the knowledge learned in the game can be transferable to a non-gaming context and this knowledge can be retained by some students"* which confirms the conclusions provided by the other sources. Despite that, Chan's research has its limitations. As with other cases, there are many ways in which a game can be designed and a lot of different methods to teach computer programming. Chan herself argues that *"there is still much left to investigate with regards to the educational effectiveness of a video game"* and points out that the results provided are limited. In addition to that, she claims that while the game had a positive impact on student performance in exams and had positive reviews, the results do not *"directly compare the educational effectiveness of the video game to more traditional methods of teaching"*. This is opposite to the previous source, where a control group is being used to gather comparable data.

2.5 Summary

The topic of using games as an educational medium is widely discussed in the scientific world. Most studies agree that games can make a subject more entertaining by allowing students to have fun while learning. From the students' perspective, this is also the most frequently mentioned good quality of educational games. Regarding to game features, most studies have showed that visual and acoustic elements positively impact a game's quality and playability. The use of storylines, teamwork, competition, social networking and more is also encouraged. A key takeaway is that a game needs to be easy to use and easy to play at the start while still featuring challenging activities at later stages to maintain interest. The approaches used to develop these games are mixed. Some have chosen to develop many games while some developed a single game. Several studies used 3D games with better graphics and some used more basic 2D games that were more focused on gameplay. A balance needs to be found between such features. Moreover, a game should have a good balance between the features that make it a game and the features that make it an educational platform. Most studies also conducted experiments either at university preparatory or high school level which reveal that games increase student performance, punctuality, consistency and retainment. Different methods of evaluation have been used to conduct these experiments including surveys that rated features of the game or even included programming questions. Some have chosen to evaluate their platforms using student grades or assignment submission times.

Despite their important contributions to the field, all of these studies have conducted their experiments using people who were voluntarily enrolled. As a consequence, they only reveal the opinion of those who are already interested in computing or programming. To understand how the general public - that is not aware of the relevant subject - can be reached and engaged, a study needs to be conducted using students that do not have programming experience or any prior desire to have any as of that moment.

2.6 Hypothesis

This research aims to confirm the hypothesis that *videogames can provide a viable alternative method to teach programming*. As an extension of this claim, this project also attempts to reveal insights regarding the perception of programming by students who have no prior disposition to computing. The hypothesis in this case is that *students without prior disposition to computing will not be interested in programming*.

3 Project Planning

3.1 Introduction

This project was scheduled to take place in a series of phases. After each phase, the progress was evaluated and the results of the phase were used in the next phase similar to a pipeline. While this may sound like a traditional (waterfall) methodology⁴, it was predominantly used to split the project into milestones and does not necessarily mean that no interventions were made in phases after they were completed. The project consisted of the following phases:

Stage	Description
Stage 1	Agree with project leader about the specifications of the project and decide on the general approach taken to implement it.
Stage 2	Develop a prototype of the game that explores different ways of visualizing mazes and how the player can control their avatar. Host a basic version of the game on AppEngine.
Stage 3	Decide and agree on the general design of the game.
Stage 4	Implement an initial version of the game with several basic activities/views and features.
Stage 5	Implement an intermediate version of the game featuring most would-have and should-have requirements.
Stage 6	Implement final version of the game featuring all would-have, should-have and most of could-have and would-have requirements.
Stage 7	Deploy and test the game's features online and offline.
Stage 8	Launch the game and publish it on Google Play Store.
Stage 9	Evaluate the game using an appropriate experiment.
Stage 10	Make conclusions on the data gathered to confirm or refute initial theory.

Figure 4 - Stages of the project

3.2 Methodology

When developing any kind of software, the methodology used is very important to guarantee the timely completion of the project and that it meets its requirements. The choice of methodology between the traditional and agile approaches is primarily dependent on how adaptable a project must be to unforeseen changes and how involved the client needs to be during the development process. This project has many characteristics that require the Agile approach. Firstly, it needs to be developed in a limited time span. Such projects often require the utilization all available time and this can be achieved using several Agile techniques such

⁴ Waterfall model: <https://airbrake.io/blog/sdlc/waterfall-model>

as timeboxing and MoSCoW prioritization. Timeboxing can ensure that the project is delivered on time and in conjunction with prioritization allows the developer to maintain their focus on what needs to be done next. These techniques also make sure that required resources are available before proceeding to the next phase and thus help to ensure the quality of any project, as seen by the following diagram:

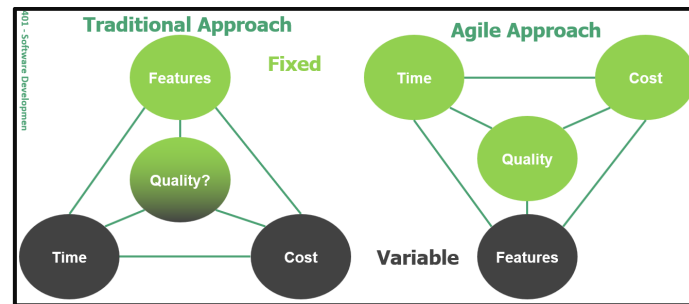


Figure 5 - Diagram comparing the impact of traditional and agile approaches on a project's quality.

Furthermore, using this approach allowed for constant feedback by my supervisor and forced me to make the appropriate decisions when necessary without wasting time and losing momentum. According to (AgileManifesto.org, 2009) the agile approach prioritizes:

1. *“Individuals and interactions over processes and tools”* – enabling focus on creating a solution that correctly interacts with the target audience rather than focusing on implementation details. This also made sure that the focus was on innovation and not on fully understanding every tool that was used.
2. *“Working software over comprehensive documentation”* – hence avoiding the creation of unnecessary documentation explaining how the product works, which in the end could become obsolete.
3. *“Customer collaboration over contract negotiation”* - thus enabling constant communication between myself and the project owner, leading to less misunderstandings and therefore less time wasted.
4. *“Responding to change over following a plan”* – allowing a quick change in direction whenever necessary. This was especially important because of a prototyping process during design and implementation.

During the implementation of the project several prototypes of the game will be planned and produced using an iterative and incremental approach. New features will be added at each iteration and the previously existing features are improved. Throughout these iterations testing is also undertaken on the completed parts of the solution, an approach that is used in Extreme Programming according to (Hneif & Ow, 2009).

For the evaluation of the project a survey will be conducted. A questionnaire created will be given to students which aims to gather information regarding their prior experiences in programming, their programming knowledge and their opinion about programming. The objective of the questionnaire is to allow a comparison of these opinions and skills before and after the game. Because of both objective and subjective questions in the questionnaire, a balance is kept between opinionated and unbiased responses. The questionnaire was delivered in paper for reasons discussed in a later chapter and included only close-ended questions (Babbie, 1990).

3.3 Requirements

Since aMazeChallenge is an educational programming game, one of the most important requirements of the game is the programming language that will be used by the players to control their in-game character. Such a language needs to have some specific characteristics:

1. It should be easy to understand and write by a beginner. (*Non-functional*)
2. It should be easy to write and run on a mobile device. (*Non-functional*)
3. It should have a reasonable performance on a mobile device. (*Functional*)
4. It should not constrict the complexity of any algorithm. (*Functional*)
5. It should be stable and available at all times. (*Non-functional*)
6. It would preferably have support by a community of developers. (*Non-functional*)

aMazeChallenge needs to visualize a maze environment (*functional*) on mobile devices with various screen sizes and this creates a major challenge in terms of usability. The graphics shown by the game need to be compatible with at least the majority of screen sizes (*non-functional*). Therefore, an appropriate interface design approach needs to be considered. In addition, the large variety of hand-held devices in the market today makes it almost impossible to create a game that works for everyone.

The multiplayer feature of the game demands hosting on an internet platform that is available to anyone in the world (*non-functional*). There are several solutions to this requirement that have distinct advantages and disadvantages. The chosen solution needs to be flexible, scalable and with reasonable performance (*non-functional*).

To get the necessary attention by young students, the game needs to feature a range of graphical and acoustic elements (*functional*). Different methods of game development need to be examined to come up with the best possible solution that produces attractive audio-visuals and is feasible within the timeframe of the project. Lastly, aMazeChallenge

should be a well-tested quality solution that does not result in any kind of inconvenience or negative experience to the player (**functional**).

3.4 Potential Solutions

3.4.1 Programming Language

Various studies that have been conducted using educational games have used a range of methods to incorporate programming. In some studies, industrial languages have been used by the participants to take part in the games or other activities. For instance, (Vinicius & James, 2016) used C++ to allow students to solve several exercises. Similarly, according to Long's study (Long, 2007), Robocode players need to learn and use Java in order to take part in the contest and create, manage and control their robots. In an analogous way, (Harris & Jovanovic, 2010) talk about a beginner-level university course that uses game programming to motivate students to learn. In this course, another commercially capable language and IDE are used – Visual Basic using Microsoft's Visual Studio. (Kiss & Arki, 2016) go as far as using C to show students how some basic games can be implemented using structured programming. In this method students were shown how a computer could simulate events such as dice throwing or card shuffling in various card games and competed in throwing dice against their own programs. The use of such commercial (text-based) languages has two main advantages:

1. Eliminates the need to shift from using other methods of programming to using text-based languages once the learner is acquainted.
2. Often provides a larger number of tools and techniques that can be used to create programs compared to other methods.

However, it also has some important disadvantages:

1. Commercial languages often have a steep learning curve especially for the complete beginner. The syntax and semantics of a text-based programming language can make a beginner feel overwhelmed and may cause low retainment rates.
2. The use of a variety of tools is often not necessary at an introductory level.
3. Writing programs in such languages requires a lot of input, which conflicts with basic usability principles related to mobile application design.

On the other hand, (Kiss & Arki, 2016) suggested that using robotics to teach programming is a viable idea. Kiss says that "*[His] experience shows the LEGO-Mindstorm is a very good tool for learning programming because the students can construct a robot with different functions and write programs without syntax error*". However, such a method has several limiting issues:

1. A large number of equipment needs to be purchased to address entire classes of participants, which may come at a high cost.
2. Students are not able to use these games after classes, at home or after specific events.

Both text-based languages and robotics games can be viable solutions to teach programming in specific scenarios. Their drawbacks are very important in this case since aMazeChallenge needs to be a game that is available at any time to players so that its outreach and effects can be maximized. In addition, it needs to be easy to play by beginners and thus feature a language that is as beginner friendly. Such languages are listed and discussed by (Karch, 2018):

1. Scratch – a free kid-friendly programming language developed by MIT, *“supplemented by tutorials, instructions for parents and a robust user community”*. Scratch uses a building-block visual interface to create a more visual experience for kids.
2. Blockly – Google’s free *“refinement”* of Scratch which is also based on building-block style code. It also features code output into different languages such as JavaScript, Python, PHP and more which makes it ideal to use for a range of audiences.
3. Alice – a free *“3D programming tool designed to teach concepts of object-oriented programming languages”* and uses a similar building-block style.

Similar games such as RoboBuilder (Wintrop & Wilensky, 2012) use such existing libraries and *“add a block-based programming language, making it easier for learners to play without prior programming experience”* (Weintrop & Wilensky, 2016). Another similar game called IPRO (Martin, et al., 2013) *“has players give directions to soccer-playing robots using a visual programming language running on a mobile device”*. Such languages can also be used to summarize taught concepts, such as variables, loops and conditionals. (Paspallis, et al., 2018) describe how during a programming event these concepts were summarized using Scratch blocks and a labyrinth game was played by the participants. The main advantages of these visual languages are evident for such games:

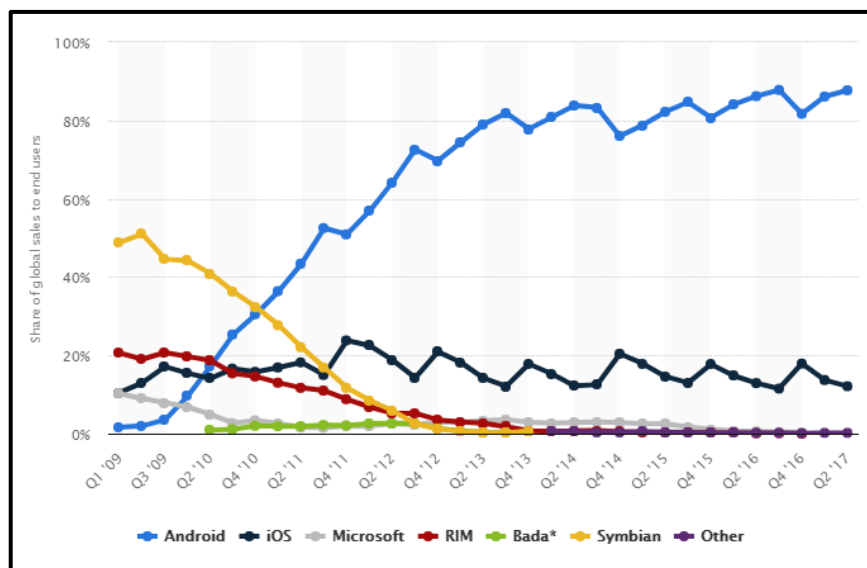
1. They are easier to write and read on mobile devices due to limited screen space and user control.
2. They are more visually appealing than text-based languages.
3. They are beginner-friendly and usually provide type safety.
4. They can easily be used to determine the sequential patterns of learners through block placements and level completion (Shih, 2017).

The choice of language results after testing different approaches prior to starting the project. An interpreter for a new language called “aMazeLang” was conceptualized but was judged as non-feasible due to time limitations and the inherent advantages and support for visual languages. A shift to JavaScript and later Blockly (which supports it) was regarded as the best approach to conclude the important choice of language selection.

3.4.2 Platforms and devices

A main requirement of the game is the development of a “mobile view”. Such a view requires targeting specific platforms and their operating systems. The obvious debate for smartphone and tablet operating systems is between Android and iOS devices. Each has its own advantages and disadvantages but the choice to develop the game for Android was directly made because of several reasons:

1. The availability of free development tools and IDEs.
2. The almost negligible cost of development and publishing to Android markets such as Google Play Store compared to Apple’s App Store.
3. The fact that Android is the most popular mobile operating system by far (Statista, 2017), allowing for larger participation:



In addition, a web-based solution was not developed because:

1. Slower processing speeds and unstable internet connections are a liability when compiling code or rendering graphics. According to (Jobe, 2013) “*native apps are still the best choice for hardware intensive apps*”.
2. An Android native application can accommodate the vast majority of devices primarily in Cyprus and ultimately the world.

3.4.3 Gameplay, Graphics and Audio

Gameplay is perhaps the most important feature of any game and this also includes educational games. (Combefis, et al., 2016) mention in their report that “*contests make the teaching of programming more attractive for students*”. Introducing competition into the gameplay was very important for aMazeChallenge because “*students [will] have the possibility to compare their abilities and learn from others*”. This pressure in turn motivates students to learn, participate and improve. The main requirements of the project dictate that players should be able to move a virtual avatar within the game to reach the exit of the maze. Such a task is achieved by writing code in a visual editor. The code is translated into runnable actions that correspondingly move the player inside the maze space, which is divided into “cells” with walls in between them that cannot be passed. As the game features different objects with which the player can interact, it is important to know how to write code to avoid or grab them. Therefore, an element of surprise or randomness is introduced to the game as these objects are spawned at random times and places inside the maze. The frequency of generation of these objects is set for each maze and can be customized for mazes created by the player.

According to (Laamarti, et al., 2014) these are some key factors that “*accelerate the move of serious games towards mass adoption*”:

- 1) User-centred software engineering – Provide the player with the necessary action when they need it depending on the context of the game.
- 2) Multimodal serious games – Different features of videogames should be incorporated such as audio and sound. They should be at the right level so that they benefit the learning experience but not distract players from the educational objectives.
- 3) Social well-being - Stimulating a feeling of virtual presence or connectedness with virtual objects (such as the avatar inside the maze).
- 4) Adaptive gaming - A serious game should adapt to a particular player’s capabilities, needs, and interests. This is done by introducing a multi-level system into the game where both beginners and experienced players can be challenged at the right level.

An example of a game with similar gameplay is Karel the Robot (Pattis, 1981), “*in which players write simple programs to control an on-screen robot as it moves around a two-dimensional world*”.

3.4.4 Online hosting

According to (Knutsson, et al., 2004) a method of hosting games on the internet is peer to peer. This is a scalable “*self-organizing, decentralized system*”. Despite being a very popular option for hosting online games, the choice of peer-to-peer is probably not the best

for a game such as aMazeChallenge. Firstly, aMazeChallenge requires that code written by players is uploaded on a server for processing. Such a task cannot take place on the player's machine as different players may run machines of different performance or connectivity and this may result to unfair advantages. Instead, the dissemination of the game's state should be avoided and a more centralized approach needs to be chosen. As games are software and -overlooking any technical differences- an application is also software, aMazeChallenge can be hosted like one. Applications are usually hosted online using either a dedicated server or a cloud server. While the dedicated server approach provides full control and possibly higher performance in terms of bandwidth it has severe drawbacks such as limitations in scalability, liability to hardware failure and maintenance costs (Mombrea, 2012). On the other hand, cloud servers have decreased performance, less storage space and give rise to privacy concerns. However, all of these are not issues that affect aMazeChallenge, because it is neither a high-bandwidth or high-storage application nor stores any sensitive data online. The choice to use AppEngine was made because it is a platform that supports Java (in which the majority of the project runs), features a highly usable control panel and is a scalable solution. AppEngine is free to use up to certain quotas and applications hosted on it are very easy to create and maintain.

3.5 Costs

Most of the software used in this project is open-source and therefore does not product an actual starting cost. If at any moment in the future the game becomes more popular, the AppEngine server can be scaled up for a set charge, however this is beyond the scope of the present report.

3.6 Tools and Techniques

During the development of aMazeChallenge several tools are used to implement and deploy the game. First and foremost, Android Studio⁵ is used to develop the client application. Android Studio is an Android development IDE and supports many useful features for the development process such as an android emulator, simulation of different device conditions and world-class code, editing debugging and performance tooling. All of the above are essential when building high quality mobile applications or games. Android Studio utilizes Java to create mobile software, so the Java SDK⁶ and JRE⁷ has to be downloaded for use throughout the development process. To allow the utilization of Google's Blockly language, the Blockly Android library⁸ is imported to compile the block structure to

⁵ Android Studio: <https://developer.android.com/studio/index.html>

⁶ Java SDK: <http://www.oracle.com/technetwork/java/javase/downloads/jdk9-downloads-3848520.html>

⁷ Java JRE: <http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html>

⁸ Blockly for Android: <https://github.com/google/blockly-android>

JavaScript. To interpret and run the JavaScript code compiled by this library another library has also been important featuring a JavaScript interpreter called Mozilla Rhino⁹. For the game's deployment, the AppEngine SDK and the Google Cloud SDK are necessary. Adobe's Fireworks provides an ideal solution to design graphics, logos and banners etc. Finally, GSON¹⁰ can be employed to parse JSON formatted strings and automatically convert them into objects.

aMazeChallenge is a game about programming, so it makes sense that it should showcase it in its code. The project needs to follow suitable software engineering techniques that make its code as organized, modular and readable as possible. The possibility that the game will be maintained and possibly expanded in the future needs to be taken into consideration at every step. Because of these reasons, the Model-View-Controller architecture is being followed. The client-server model is also important because it will allow the expansion of the project into multiple applications in the future and distribute the processing across multiple devices.

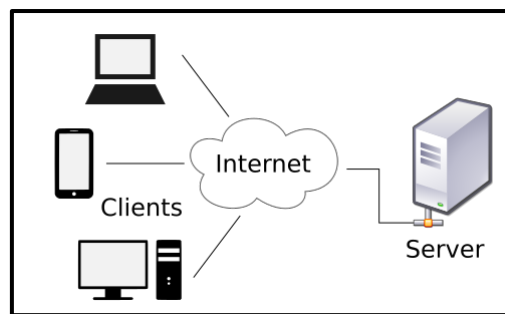


Figure 6 - Client-Server Architecture

aMazeChallenge will be mainly tested iteratively because of limited time but a period will be left at the end to test the system as a whole. After each iteration the added features are tested after they are implemented. If something is omitted or found to malfunction it can be fixed at any moment during the implementation without disrupting the project. The non-functional requirements can be tested at the very end during system and acceptance testing.

For the evaluation of the game, a survey will be conducted among students of a local high school. The questions of this survey should indicate any differences between the opinions and skills of the students related to programming before and after playing the game. Different kinds of graphical representations will be used to demonstrate these results.

⁹ Mozilla Rhino: <https://developer.mozilla.org/en-US/docs/Mozilla/Projects/Rhino>

¹⁰ GSON: <https://github.com/google/gson>

3.7 Legal and Ethical Issues

3.7.1 Electronic personal data

According to Cypriot and European directives and laws, individuals have specific rights concerning their personal data (European Union, 1995). aMazeChallenge collects a relatively small amount of user information: names and e-mail addresses. The collection of this data is essential to provide a personalized experience and to identify players. The data is stored securely and according to the game's privacy policy it may only be used for research purposes. In some cases, an electronic questionnaire can be provided which does not collect any personal data.

3.7.2 Questionnaire personal data

The paper-based questionnaire given to students during the study will collect some personal data including their date of birth, gender and experiences in programming. Such data does not identify real persons at any moment after its collection and will be presented in a summative approach. The project's privacy policy can be found in **Appendix 1**.

3.7.3 Other issues

Because the game is primarily targeted to an audience predominantly under eighteen years old, special care should be taken so that none of the players are exposed to harmful situations or unpleasant material. Players who are under the age of eighteen need to have their parent's or guardian's consent to participate in the game and its related questionnaire. Such as permission can be granted on-site.

3.8 Summary

The overall project plan has specified distinct functional and non-functional requirements of the game, has analysed different approaches in meeting these requirements along with their use in past related work and any advantages or disadvantages they may have. The ethical and legal issues regarding the project have been assessed and addressed. Specific tools and techniques will be used to develop, test and evaluate the game. Corrections can be made during the design and implementation processes but an overall congruence with the initial plan needs to be maintained.

4 Design

4.1 Introduction

In this chapter the game's requirements will be broken into technical categories and be analysed. Potential solutions to these requirements will be examined and the most relevant and useful one will be chosen and illustrated with the aid of either diagrams, pseudocode or other forms of presentation.

4.2 System Characteristics

The system does not require a lot of storage either as a form of database or in the device's file system. Apart from the game's installed files, additional files could include maze configurations and other user preferences. The types of data stored online by the game in the form of a database will be limited, although the data itself may reach high volume in the future. Such data is mainly related to the game and survey and does not require any particular security attention although it should be inaccessible by anyone except the administrator.

The central design challenge is how to best represent mazes, player movements and how to incorporate these elements into a continuous game loop. Because of a potential expansion in the future, the design needs to be modular and flexible to allow for such changes. In addition, the game should be kept as simple as possible and guide the player through the steps required to play it. For example, a careful consideration of the game's flow will make it far more likely that participants will be correctly educated before playing the game. Lastly, certain principles need to be followed to ensure the design of a robust user interface that is both user-friendly and visually attractive.

4.3 System Architecture

4.3.1 Application level

On the application level, aMazeChallenge is broken down into several components. A class will be created for each of these components to represent a real-world system. The organization of these classes will take place in a manner that allows the differentiation of presentation and logic. One method of achieving this is the Model-View-Controller architecture, which separates data, presentation and control layers. These layers interact by sending data to each other at runtime. For example, when a list is shown, the model layer will send data to a view so that it is presented to the user and the controller assigned to this view will determine how the system behaves when an item in the list is selected. Such an architecture has several advantages according to (Jithin, 2016). Firstly, it separates the

implementation of data representation from the way the data is presented. This provides the “*ability to provide multiple views*”, for example displaying a set of data as a list or a dropdown box. Secondly, modifications made to one model do not affect the entire architecture and any created classes are completely independent from each other. These classes will therefore be kept short and trivial to understand, increasing the code’s readability while also making it easier to debug. An added advantage is that this method also works better with asynchronous calls compared to a regular mixed layer implementation. Despite the increased complexity of this architecture and the fact that in this specific scenario it does not offer any real time reduction in the development process, the described advantages will certainly help in the long run, when code needs to be read, understood, maintained, expanded and most likely debugged.

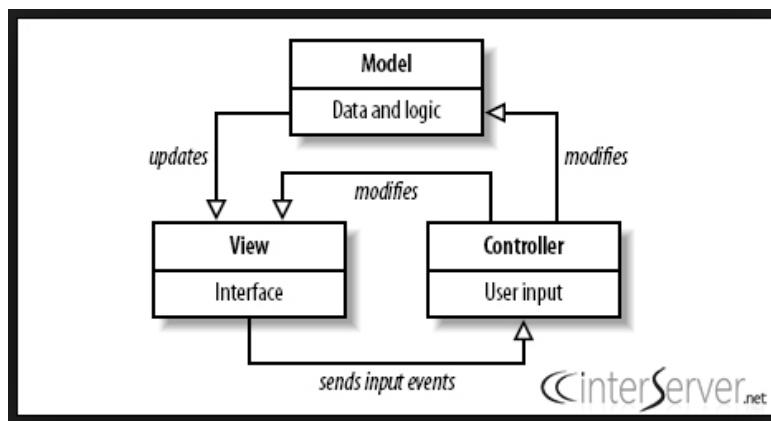


Figure 7 - The MVC architecture

4.3.2 System level

The system-level architecture is concerned predominantly with how different components of the system communicate. The system comprises of several components as shown below:

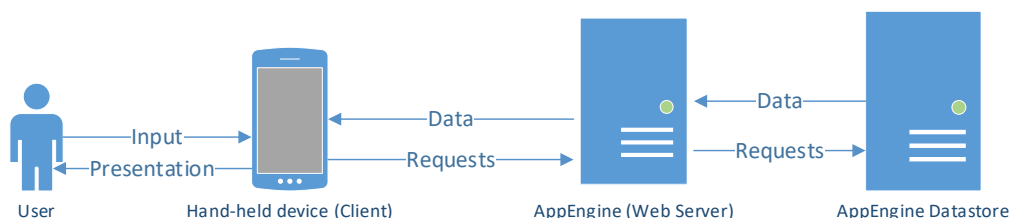


Figure 8 - The system architecture based on the client-server model

When the user interacts with their device their input is translated into a request. A request may include data that will be stored on the server, such as the player’s code. The request is then sent by the device through the internet (omitted) to a web server, which hosts

different web services. According to the behavior of each web service, the server will determine if a connection to the data store needs to be made. If this is the case, the web server will query the data store with another request, which returns the necessary data. When the data is received, the web server responds the initial client request by relaying that data into a specific format that can be understood by the client device and presented to the player.

4.4 System Design

4.4.1 Application flow

The following diagram describes the flow of the game's menu screen and user interactions with it:

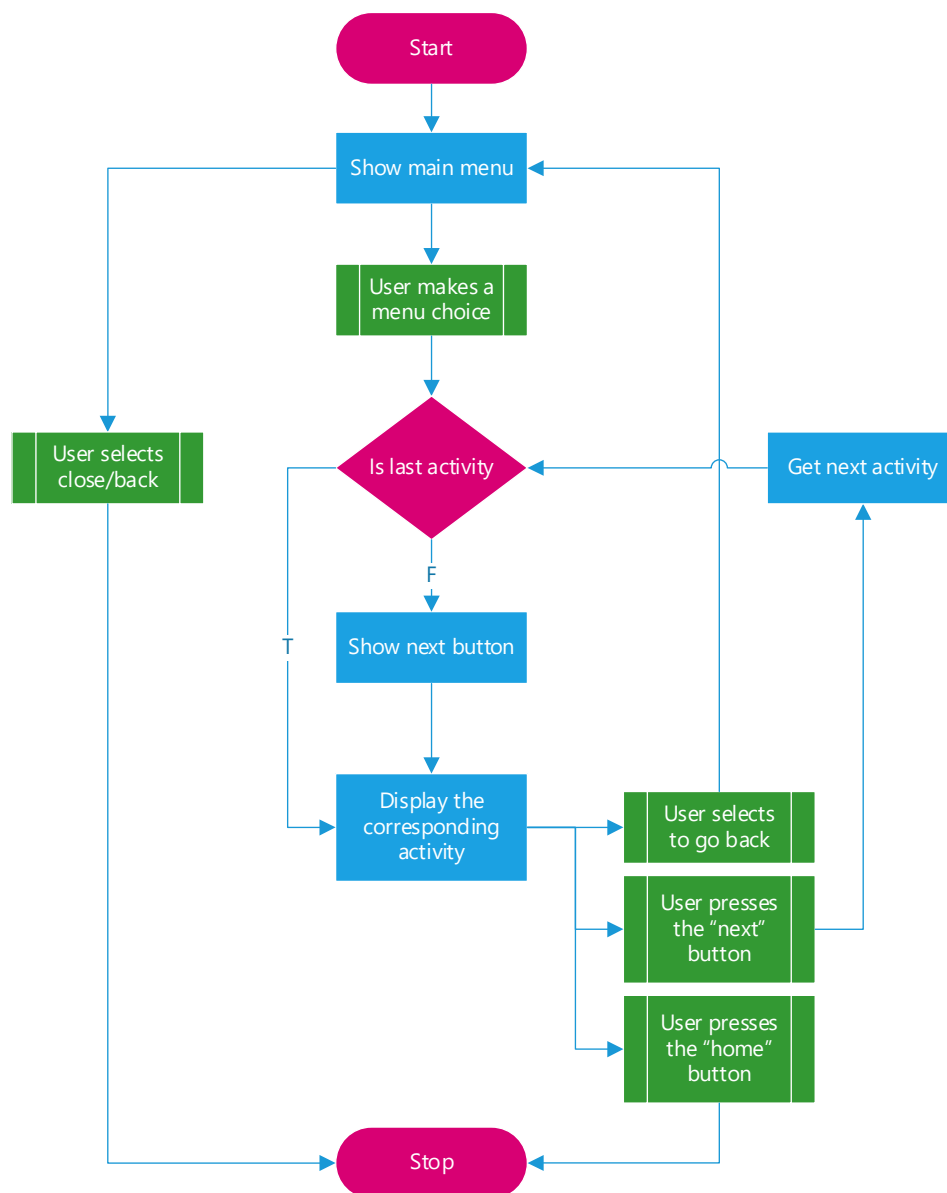


Figure 9 - A flowchart representing the game's flow from the menu to different views.

The main menu needs to provide a flow between different views of the game. Using this method new players will not need to retract back to the menu screen to find the next step in the game, allowing them to easily navigate through the application and saving them valuable time which is important for any competitions that take place. However, the option to go back to the main menu is still available in every activity so that compatibility is maintained with current standards in navigation. The game's flow between activities is roughly described in the following list:

1. Learning – Teaches basic programming concepts and the game's rules and mechanics.
2. Personalization – Allows the player to select personification options.
3. Code Editor – provides an environment where the player can write their code.
4. Training – Lets the player to execute their code in training scenarios and see how their avatar behaves.
5. Play online – Registers the player on a selected challenge and displays their progress in an online game.
6. Maze Designer – Allows the player to create their own mazes and add them to training.
7. About – Displays information about the game.

In addition to this flow between activities, the main screen will provide options for the selection of two preferences:

1. Sound and vibration;
2. Language.

Both of these options will not cause any changes in the game's view flow.

4.4.2 Game rules and mechanics

The online game will be turn-based, meaning that the players will be in order of play according to their code submission time. For example, the player who submits their code first will be the first to play a turn. Each player's code will be executed once per turn in a continuous loop of turns until either all players have lost or won. When a player is registered for a challenge (an online game) by submitting their code, their state depends on the number of available slots for that challenge. If there are available slots on the selected challenge the player will be admitted. If there are no slots available the player will enter a queue until another player loses or wins. The diagram in the next page shows how the game handles multiple players.

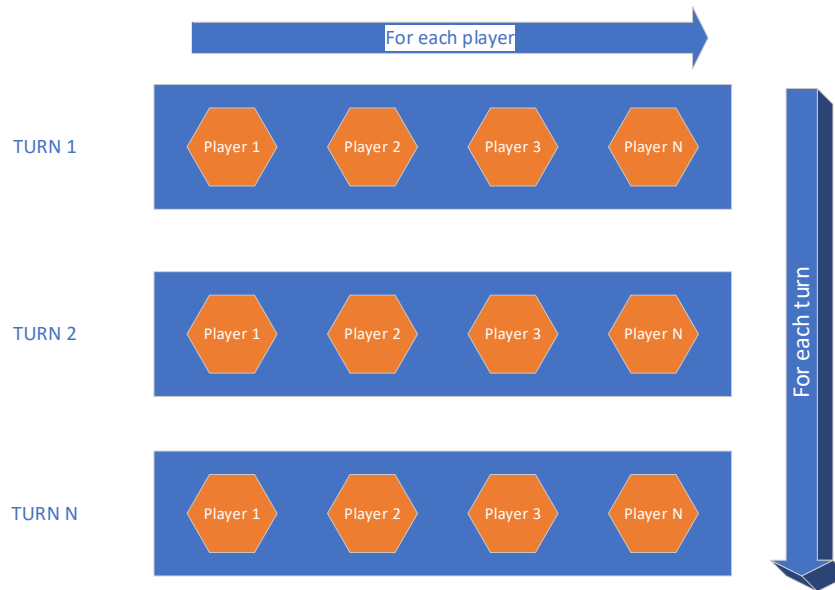


Figure 10 - A diagram showing how players' turns are executed.

Depending on the challenge's configuration, two or more players can be at the same position simultaneously, but this does not affect the order of turns.

Furthermore, the game introduces two properties that have a direct impact to a player's survival or chances of winning. Firstly, each player has a health value which is initially set to one hundred. This value is not the maximum health and players can increase it up to a maximum value by picking several items. Similarly, their health can be also decreased by other items inside the maze, until it reaches zero – which means that they lost and are removed from the challenge. Several items that have been chosen to affect the player's health are:

- 🍎 Fruits – a variety of fruits each of which positively affects a player's health by a different value.
- 💣 Bombs – a timed object that explodes after a few turns and causes significant amount of damage to a player.



The second property affecting the player's success is points. Points are given as a reward for catching the following types of objects:

- 📦 Presents – a gift box that gives a player a large number of points.
- 🪙 Coins – a variety of coins each of which gives a different number of points to the player.

Players that complete the maze first are the winners of the challenge. In specific contests however, points may be used to rank players who have not managed to complete a challenge or even change the order of those that have completed it (if the time of completion

is close enough). Such details of player ranking are yet to be determined since the first release of the game is planned as a prototype and will hopefully reveal the best way to rank the players.

As mentioned above, each player normally gets one move in the maze per turn. However, a couple of in-game items also affect the number of moves made per turn. More specifically, objects can cause a player to move faster in the maze for an amount of time, while others can cause them to move slower or not at all:

-  Double moves – a player can make two moves each turn, essentially doubling their speed for a limited amount of turns.
-  Traps – causes the player to lose their move for a limited amount of turns which makes them appear stagnated at the same position (as if they fell into a trap).

The speed/move variations caused by these objects may not seem so important. However, they provide a random element of a player's speed to the game. Two identical algorithms will finish the game at the exact same time (although one player will finish first depending on the order of code submission). Therefore, by introducing these objects a player's completion time will vary significantly depending on how many objects of this kind they have interacted with and how well they have managed to avoid “penalty” objects or catch “reward” objects.

4.4.3 State representation

4.4.3.1 Game (Controller) state

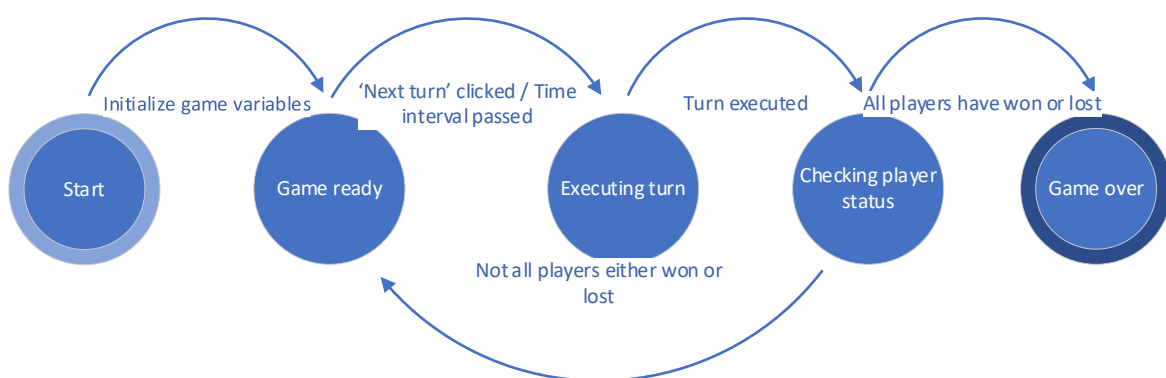


Figure 11 - A state transition diagram representing the game state.

Figure 12 represents a simplification of the states of a game object and the transitions between them. This diagram applies for both tutorial and online games. When a game is created certain variables are initialized - which will be discussed below. The game then

enters a state of readiness and waits for the player to press 'Next' in the tutorial or the time interval to expire in both tutorial and the online game. When this happens, a turn is executed for each player in the game. When all player's turns have finished executing, the game enters the 'checking player status' state where it checks if all of the players have either won or lost. If this is the case, the game stops and appropriate messages/events occur in the application. Otherwise, the game returns to a state of readiness and waits for the 'next' button to be pressed or the time interval to pass.

4.4.3.2 Picked object state

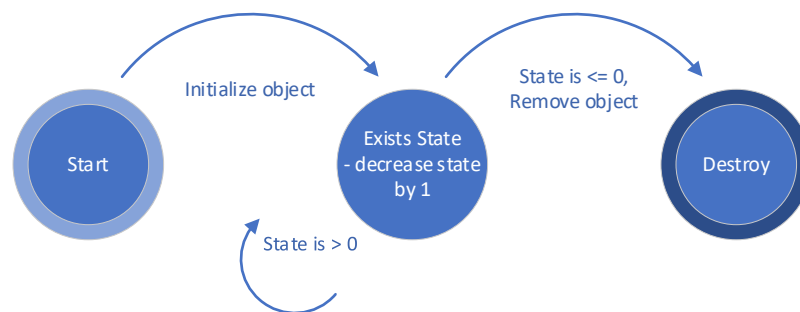


Figure 12 - A state transition diagram showing the state of picked objects

NOTE: The term *pickable*, although not an English language term, refers to an in-game object that can be picked by the player or have an effect on the player's properties.

The state of picked object is initiated by the game's controller during the "initialize game variables" transition. When a "pickable" object is created its state is initialized to an integer number which corresponds to the number of rounds that it should exist. At the passing of each round this number is lowered by one and if this number reaches zero, the object is removed from the game.

The state of bombs has some unique characteristics in state transition due to the nature of these objects. Bombs feature a time-delay before exploding and a change of their texture also warns players when they are about to explode. To achieve this peculiarity, the states of bombs transition differently from other pickable objects. Figure 14 below shows the transitioning between different states to achieve this change in texture as well as the explosion of the bomb.

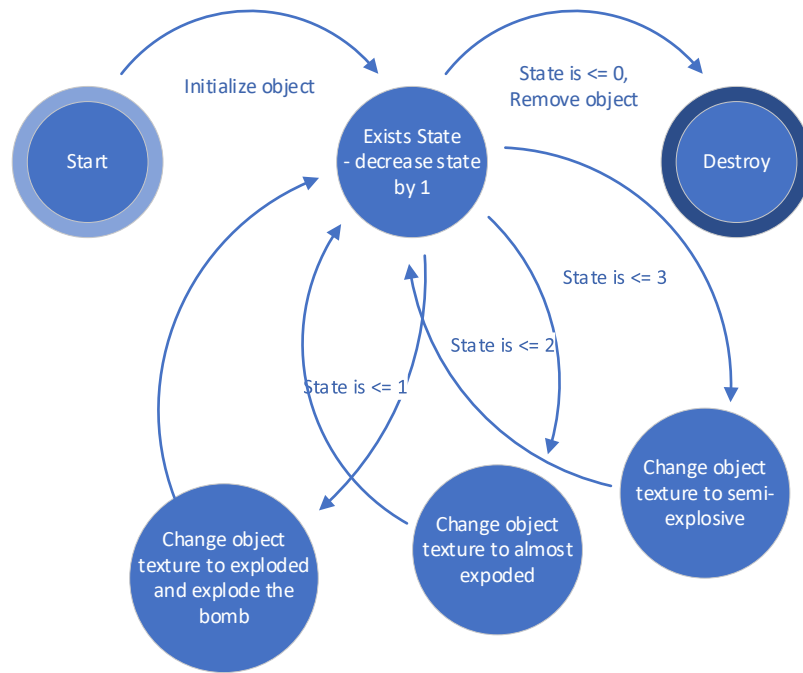


Figure 13 - A state transition diagram depicting bomb states and the transitions between them

4.4.3.3 Code editor and Static Checker states

The diagram in **Appendix 2** shows how the code editor and static checker's states are connected to provide the player with editing and checking features. After a player writes their code and attempts to compile it, the static checker will change its state depending on the outcome of its processes. If the code passes these inspections, it will be considered valid and will be compiled, otherwise an error will be displayed.

4.4.4 Data Structures and Algorithms

4.4.4.1 Data Structures

The classes of the project will be mainly categorized in a package called model. Such classes provide correct code encapsulation and other object-oriented features such as polymorphism and inheritance. Firstly, some simple enumerator classes will be used to store data related to different objects in the game. Some examples of these objects are:

- The player's colour and icon;
- Audio and Images;
- Challenge Difficulty;
- Direction and position;
- Languages.

The following fragment of a class diagram accompanied by a CRC card provide an example of how simple classes like the Colour enumerator class could be implemented:

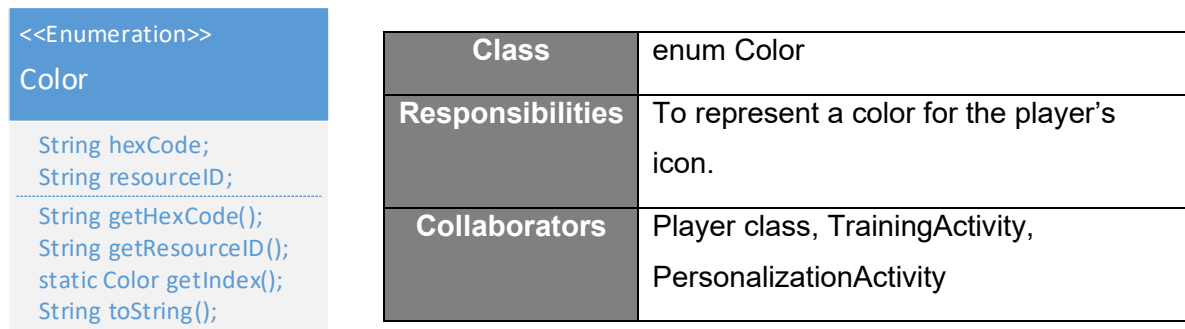


Figure 14 - Fragment of a class diagram and CRC card showing how color could be represented in the game.

One of the most important classes in the game is the “Game” class, which is responsible for keeping the maze’s layout information, lists and queues of players, the pickable objects inside the maze and the game’s state and counters. It will also provide important management functions to add and remove players, reset their statistics, queue them into a challenge, check their positions and directions, retrieve the objects inside the maze and more. The game class will also be responsible for storing event listeners, which will be detached from specific activities according to the MVC architecture.

Another very significant class is the “Challenge” class, which stores information related to each challenge. A challenge can be regarded as a specific instance of a game that contains a particular structure for the maze as well as its configuration. For example, each challenge must have a name and description, and must store a representation of the maze environment. Each challenge object should also store the challenge’s starting and end time for contests, as well as some configuration:

1. If the challenge can be played multiple times by the same player.
2. If players can join after it has started.
3. If players can be at the same position simultaneously.
4. If a questionnaire will be shown at the end of each challenge.

In addition, some graphical enumerators will be stored for each class such as for the background audio and image, the lines of the walls and the challenge’s difficulty. The challenge class is also responsible for calculating the number of objects spawned in the maze according to each pickable object type’s intensity rating.

The grid class is used to store information on a maze’s structure. The maze structure will be represented using a hexadecimal string in which each digit represents a type of cell

with specific walls. For example, a hexadecimal value of 9 which is translated into 1001 in binary will represent a cell which has a right and upper side wall. The following diagram demonstrates how each cell can have up to four walls using this technique. Assume a hex character “E”, which is translated to 1110 in binary. When each digit of this sequence is checked using bitwise AND operations with a mask (e.g 0100 for the second digit), a side of the cell can be said to have a wall if the resulting value is one.

Up	Down	Left	Right
1	1	1	0

Value Mask

1110 & 0001 = 0000 (No right wall)

1110 & 0010 = 0010 (Left wall)

1110 & 0100 = 0100 (Down wall)

1110 & 1000 = 1000 (Up wall)

Results to:



The position and direction of players is very important for the game. The position of each player represents the cell in which they are in the game, using x and y coordinates. The direction of a player is based on a natural compass, with north facing up. The direction class will have to support turning of the players, which will also be counter as one valid move. As such, methods like “turnClockwise” and “turnCounterclockwise” will have to be implemented that change the players facing direction according to their current direction (i.e. if a player is facing North and a turn counter-clockwise move is made the player will be facing East). Similarly, the position class will have to implement functions such as “moveNorth” etc. or moving in the direction the player is currently facing at. As a rule of the game not mentioned before, the following are the player’s valid moves:

1. Move forward (in the currently facing direction);
2. Turn clockwise;
3. Turn counter-clockwise;

Some of these moves can be invalid at certain contexts. For example, if the player is on the right side of the maze and is facing west they cannot move right. However, this move is permitted by the game and is part of teaching players how to write correct code that manages such situations.

A class responsible for compiling the player's code needs to be created, which will be communicating with the Rhino library. The player's code will be sent from an activity to an instance of this class along with some other information and will then be compiled. Preferably this compilation process will take place asynchronously so that certain time limits can be enforced. If the code takes too long to compile, the game should consider the code to be invalid and stop the compilation which will cause the player to lose their turn. Because of interpretation taking place in this class, an instance of itself will be sent to JavaScript code.

After compilation, a controller class is called within an activity to play the game. The controller class needs to consist of static functions only and make certain operations on the provided games and challenges. These games and challenges will be passed as parameters to the functions. The controller class is responsible for running a given game. For example, given a game, a `makeMove()` method will go through each player in the maze and apply their next move. In addition to moving players in the maze, this class is responsible for spawning the pickable objects, applying their effects and keeping count of any variables that affect the gameplay such as any players with double moves along with the remaining rounds of the effect. The logic of finding if someone has reached the finish position should also be implemented in this class.

4.4.4.2 Algorithms

Several algorithms will have to be employed to test the game. Such algorithms will be implemented by types of `MazeSolvers` which will preferably extend a common interface. Such examples of algorithms include:

1. Random Maze Solver – an algorithm that generates a number from zero to two which corresponds to a move (move forward, turn clockwise, turn counter-clockwise). That randomly generated move is then applied by the controller.
2. Left Wall Follower – an algorithm that implements the left-hand rule, following the left wall until the end is reached.
3. Right Wall Follower – the opposite of the left wall follower, implementing the right-hand rule and following the right wall until the end is reached.

These algorithms will be available to players as examples. Their complexity is minimal and the objective is to teach learners how to create basic code structure. The random maze solver is expected to take much longer to complete even a small maze compared to the

other two algorithms. However, none of these algorithms will be able to solve non-perfect¹¹ mazes in a realistic time (or perhaps never).

4.4.5 Web Services / API

Several web services need to be developed as discussed in the planning section so that players are able to register for a challenge, upload their code and receive the maze information to be displayed on their devices. The API of the game needs to be available to anyone but also provide security and authorization/authentication for certain functions such as creating new mazes, viewing the player information etc. This is a feature incorporated into Java servlets, a technology that allows easy creation and hosting of web services. Other alternatives to this approach include the use of other server-side languages such as PHP or C#, but this would be time consuming since neither have a built-in security system and do not support Java-based tools such as GSON (a form of object-relational mapping) and Rhino. Hosted web services will provide data to client devices in a specific form, most probably JSON. Parsing and creating files of this data transfer method is made trivial by the use of GSON, a Java library that converts JSON text to and from Java objects. Oppositely, the alternative of XML is harder to parse and produces larger text which is not beneficial in terms of bandwidth. The API should also interact with the AppEngine data store to retrieve data that will be relayed to the client devices.

4.4.6 Datastore

The AppEngine data store provides a rapid development alternative to other data hosting solutions such as MySQL and MSSQL. Custom database hosting has additional costs and requires the design and maintenance of entire databases. Even though these databases offer far greater customizability compared to AppEngine's Datastore, this advantage does not outweigh the fact that they take longer to set up and more effort to maintain. Hence the choice of datastore is selected. Several entities will be stored:

1. Challenges – the information related to online challenges such as names, descriptions, starting and finishing times as well as the maze grids.
2. Players – the player information such as position, direction, color and the player's code.

The structure of these objects corresponds to the class structure in Java code. A tool called Objectify¹² offers quick interfacing with the Google Cloud Datastore objects and translates them into Java objects.

¹¹ A non-perfect maze is a type of maze in which not all cells are connected by a wall.

¹² Objectify - <https://github.com/objectify/objectify>

4.4.7 File system

The file system of client devices will be used to store certain files that are saved by players. For example, players may be able to save their code and load it later. This is facilitated by saving an XML representation of the blocks in the editor. Such a file will have a name with a specific format so that it is recognized and read by the client device as a player code. Similarly, players should be able to design their own mazes and save them locally. This requires a similar style of saving files on the client's device. The files will be stored on the game's installed directory (typically located at /data/data/domain_name/files). Maze files will be stored using JSON format using a "playerMaze_" prefix while code files will be stored in XML and a "playerCode_" prefix.

4.5 User Interface Design

The development of applications and games for portable devices such as phones and tablets is faced with a set of challenges created by screen size, graphics scaling and usability issues. According to (Nielsen, 1995), there are ten usability heuristics for UIs that need to be followed. The UI of the game, including the maze representation, object movements and player objectives need to be designed in such a way that there is a direct match between what is represented in the game and real-world mazes. The status of the game should be visible to the user and appropriate feedback should be given where necessary. Consistency needs to be upheld across the game in terms of colour schemes, patterns, and control. Another important factor that affects the usability of UIs that is specifically important for mobile devices is minimalist design. Because of reduced screen size, information irrelevant to the player's cause should not be shown. Instead, the player should be able to recognize that a previous action they performed is linked with an action they have to take next. Unlike desktop computers, mobile devices and tablets are used in various contexts. Therefore, actions should require as less interactions as possible with the device to minimize the chance of error. Navigation should also be kept as simple and direct as possible, following the activity flow discussed above. This allows players to find what they need easier and faster. Finally, a visually appealing and organized interface is essential to attract the attention of young students. **Appendix 3** includes a number of screenshots of the designed user interfaces as taken from within the first Android Studio layout files.

4.6 Summary

Although the design decisions made should suffice to meet the set requirements, changes may be made to the way it is structured. Important considerations have been made regarding the architecture, structure and user interface of the application and the necessary tools, techniques, platforms and frameworks to implement it have been recognized.

5 Implementation

5.1 Introduction

The implementation of this project has taken place for roughly ten months. During this time, me and Dr Nearchos Paspallis -the project leader and supervisor- have co-jointly developed the game and its underlying system. I have personally worked on most of the Android application while Dr Paspallis created the back-end AppEngine interface and helped structure the project as a whole. In this section I am going to discuss the inner workings of the system, how it was implemented, the techniques and tools that have been used and any changes that had to be made from the design of the system or any initial plans. The implementation of this project can be found on GitHub¹³.

5.2 Programming standards and conventions

5.2.1 Language Conventions

(Sun Microsystems, 1997) outline the Java conventions and the reasons why it should be used. Sun Microsystems say that *“80% of the lifetime cost of a piece of software goes to maintenance”*, which is usually done by someone else than the original author of the code. Using code conventions for the project’s primary language ensures the code is kept readable and maintainable for a future developer or the owner of this project.

5.2.2 Readability, reusability, maintainability and expandability

aMazeChallenge was developed with a mindset that expansions or improvements would be made in the future. Creating a perfect product is extremely rare in the case of software and often new patches or entire versions are released. I personally regard readability, reusability maintainability and expandability to be extremely important when developing any software and I have tried to follow these values to ensure that any other person can continue my legacy in this project. The use of the MVC architecture plays a vital role in this approach. In addition, the use of packages to organize similar or interlocking classes makes code easier to find. As a consequence of Java being an object-oriented language, principles such as inheritance, polymorphism and abstraction, encapsulation and reflection have been used to their fullest extent. Furthermore, the use of design patterns in several cases applies reusable solutions to different sets of problems.

¹³ GitHub Repository: <https://github.com/nearchos/aMazeChallenge>

5.3 Client-side

The files of this project are separated into three different locations. The code for the Android and AppEngine sub-projects is located in each sub-project's directory. Common code between the two sub-projects can be found under the "common" folder.

5.3.1 Maze and player visualization

In order to provide a view that visualizes the player's position inside the maze arena a custom view has been implemented called "GameView". This view extends the View superclass which provides a method called "onDraw()". The onDraw() method is responsible for drawing the view to the devices viewport. In this specific view, a canvas is being used to draw the maze. Firstly, the background of the maze is drawn. This background may differ for each maze, which is why the enumerating class BackgroundImage is being used. The onDraw() method iterates through each cell by visiting all rows and then columns in each row. The cell is drawn by determining its wall configuration as described in section 4.4.4.1 and lines are drawn at each side if there is a wall. After drawing the maze grid, the onDraw() method paints the starting and target positions by highlighting them as light red and light green respectively. This is done by painting a square on top of the maze grid and an extra set of walls that are overlaid on top of the new square. Calculations on the size of each cell are intuitive and are made by using the device's viewport width and height. After painting these positions, the method iterates through the list of pickable objects for this maze and draws them at the centre of each cell. Finally, the map of players is painted in the same way. The player is drawn as a triangle on the screen which means that the drawPlayer() method must take into account the player's direction so that it is drawn in the correct way. In addition, this method checks if the player being drawn is the player of the device and places a "You" tag to make them more visible. The code for the onDraw() method in GameView can be found in **Appendix 4**.

5.3.2 Interacting with Blockly code

Code writing takes place in BlocklyActivity. This activity utilizes the Blockly for Android library and extends AbstractBlocklyActivity. By doing so, the activity is allowed to override some important methods regarding XML code generation from the inserted blocks. Furthermore, some methods allow retrieving the XML representation of the inserted blocks, which can later be saved and loaded from files. BlocklyActivity features its own action bar which displays default buttons that can be implemented to support such actions. Furthermore, an options menu can be created, but was omitted at later stages of the project because it was deemed unnecessary. In this activity, a workspace allows players to place blocks that can be obtained from the toolboxes on the left as shown in **Appendix 5**.

5.3.2.1 Code editing

Code can be edited by placing blocks in the workspace. To link the blocks together is intuitive and is done as part of the Blockly library. Placing a valid block under another highlights a link between them and joins them together. Blockly contains built-in type safety and does not allow invalid types to be assigned or placed at specific places. For example, a string literal cannot be placed as a condition in a while loop. For aMazeChallenge, players have to use two functions called `Initialize` and `Run`. The `Initialize` function runs its code once at the start of the game and can be therefore used to initialize values. The `Run` function behaves like a built-in loop (although the loop itself is not shown) and runs its code every time the player makes a turn. In this way, variables can be saved after a turn has passed and be retrieved at the next turn. The inclusion of the `Run` function is mandatory but `Initialize` can be omitted.

5.3.2.2 Code checking

After writing their code, players can compile it by pressing the back button or the next button on the top right. Before compiling the game will statically check the code for obvious errors. Although a large number of programming mistakes may not be detected at all, this static checking method provides a warning for beginners who have made very obvious mistakes that will cause them to lose turns unnecessarily. Such errors are described in **Appendix 6**, along with an example implementation of a checker.

5.3.2.3 Code compilation, processing and running

When the code passes checking, it is translated by the built-in Blockly compiler from XML to JavaScript. This process is not enough to achieve the functionality necessary for aMazeChallenge. Special processing needs to take place so that code placed by the players inside the `Run` function is executed at each turn and so that variables created in `Initialize` remain persistent throughout the game. This is done by injecting JavaScript code into the original code compiled by the Blockly tools. The method `processCode()`, in **Appendix 7** shows how this process is implemented. By doing this injection, code in Java and JavaScript can be interchangeably called from either of the languages to store variables and perform operations. After this processing is complete the code is either executed locally or sent to the server in case an online game is being played. In both cases, the interpreter responsible for running the code is Mozilla's Rhino, through the `InterpretedMazeSolverExecutor` class. This class is used to avoid obstructions in the user interface if the code compilation takes too long. This is achieved by using an asynchronous task, as shown in **Appendix 8**.

5.3.2.4 Built-in (sample) algorithms

The built-in algorithms have been created to support the learning process of beginners. These are some basic algorithms that have been discussed in section 4.4.4.2. In this way, beginners can become more acquainted on how to structure their code correctly and the correct mindset can be set in terms of programmatic thinking. In addition, these algorithms provide a way for those with no programming skills to compete in the game. They can be loaded from `BlocklyActivity` by using the action bar icon, edited and saved as a new algorithm. **Appendix 9** shows an example of a sample algorithm, the left wall follower.

5.3.3 Maze Designer and types of mazes

`aMazeChallenge` supports generation and design of three types of mazes, as enumerated in `Algorithm`:

1. Single solution (simply connected mazes with a single valid path).
2. Many solutions (mazes with multiple valid paths).
3. Sparse (mazes with a low number of walls).
4. Empty (mazes without any walls).

The use of different mazes aims to:

1. Give a start in programming to complete beginners by only using very simple commands (such as move forward) in empty mazes.
2. Increase the difficulty level for more experienced programmers using sparse or many solution mazes, where the basic algorithms do not work.

The Maze Designer (`MazeDesignerActivity`) allows a player to create their own custom mazes using lists of given background images and audio, wall colours, custom start/finish positions and more. In addition, players can edit the mazes by long-clicking in `TrainingActivity`. **Appendix 10** shows the layout of the Maze designer and some generated mazes.

5.3.4 Training Mode and maze controls

The tutorial mode displays the player's avatar and colour and allows them to select a maze to play as practice. When the player selects a maze, they are presented with a visualization of the maze in `GameActivity`. By pressing the next button, the player is shown how the code they have written behaves in a single game turn. The player's value for health is displayed using a progress bar that increases or decreases respective to the value. The points of the player are also displayed. The game can be played continuously by pressing the play button and selecting the interval between each turn. Finally, the reset button resets the game.

5.3.5 Gamification of maze-solving

aMazeChallenge gamifies the process of maze-solving in an attempt to teach programming. For that reason, some interesting objects were added to the game such as bombs, traps, coins, fruits and more to increase the intrigue of the players and add randomness to the values discussed above. The behaviour of the game for these objects is implemented in the RuntimeController class (handlePickableState() and applyPlayerMove()), fragments of which are present in **Appendix 11** along with a sample representation of a challenge in JSON.

5.3.6 Personalization

Personalization options in such a game provide players with ways to set up a personalized game experience. The options provided in PersonalizationActivity are simple but may be expanded in the future. Players can change their name and e-mail account as well as their icon colour and avatar image. Sliding selectors allow players to quickly find their desired colours and images by swiping through the lists. Implementations for these selectors are shown in **Appendix 12**.

5.3.7 Learning

The learning process is guided through several steps that aim to instruct the player on:

1. The rules and mechanics of the game;
2. How to navigate in the game;
3. How to create, edit and load code;
4. Basic programming principles;
5. How to play online;

These steps have been implemented in HelpActivity, which features a ViewPager with a list of HTML pages. These pages have been styled to be learner-friendly and to provide a positive first experience for players. Depending on the language selected, the corresponding help pages are displayed. Screenshots of some help pages can be found in **Appendix 13**.

5.3.8 Online Mode using a Web API

aMazeChallenge communicates with the server using HTTP requests to retrieve data regarding challenges and game state and to post player data and participation in challenges. To achieve this, asynchronous tasks are being used, which decouple these requests from the main thread to avoid stalling the application's user interface. An example of such a task is the SubmitCodeAsyncTask which is a private class and can be found in OnlineGameActivity. This specific task utilizes the web API provided by the AppEngine server to submit a player's code to their selected challenge.

5.3.9 Main menu and user preferences

The game's main menu screen is the first view that is displayed to players when they launch aMazeChallenge. The main screen needs to showcase the features of the entire application without causing an information overload to the player. In addition, it allows the player to select their intended action with minimal input.

The game also provides several settings that can be changed by the player to configure the application according to their preferences. These settings can be changed using the buttons in the main menu near the top:



Each of these settings opens a dialog which shows a list of possible options for that setting.

5.3.10 Questionnaires

As a way to get feedback from players about the game, aMazeChallenge employs a built-in electronic questionnaire. This questionnaire can be displayed after each challenge that has its `hasQuestionnaire` attribute set to true. **Appendix 14** shows a sample challenge configuration file represented using JSON and a screenshot of the questionnaire. The feedback received by the questionnaires is sent to the server and stored. It can then be processed and analysed to evaluate the game.

5.3.11 User interface

According to the requirements of the application and design criteria, the implemented user interfaces have been created using the following components:

1. Dialogs, ViewPagers, Spinners and Lists that minimize the amount of input necessary.
2. Saturated colours to attract attention from young audiences.
3. Toasts and Snackbars to provide information to the user in a minimalist way.
4. A generally trivial navigation scheme between activities.
5. Custom view for maze representation and animated avatars as well as personalization ViewPagers.

In addition, material design¹⁴ principles were followed to create meaningful visual elements. Several screenshots can be found in **Appendix 15** but also in other appendices where screenshots can be found.

¹⁴ Material design: <https://material.io/guidelines/>

5.4 Server-side

The server-side implemented by Dr Nearchos Paspallis utilizes the AppEngine platform and its datastore to provide web and data hosting for any distributed application. aMazeChallenge uses several web-based API endpoints as Java Servlets to achieve its functionality. The first of these endpoints is encountered when the user is presented with a list of challenges in OnlineGameActivity. The game acquires this list by making an HTTP connection to the ChallengesServlet, which queries that data store and responds with a list of the available challenges and their properties. In addition, JoinServlet is called when players register for a selected challenge. This servlet requires users to provide a set of data and makes the necessary checks to ensure that it was provided:

```
final Vector<String> errors = new Vector<>();
if(magic == null || magic.isEmpty()) {
    errors.add("Missing or empty 'magic' parameter");
} else if(!checkMagic(magic)) {
    errors.add("Invalid 'magic' parameter");
} else if(email == null || email.isEmpty()) {
    errors.add("Missing or empty 'email' parameter");
} else if(!isValidEmailAddress(email)) {
    errors.add("Invalid 'email' parameter - must be a valid email: " + email);
} else if(name == null || name.isEmpty()) {
    errors.add("Missing or empty 'name' parameter");
} else if(colorName == null || colorName.isEmpty()) {
    errors.add("Missing or empty 'color' parameter");
} else if(iconName == null || iconName.isEmpty()) {
    errors.add("Missing or empty 'icon' parameter");
} else if(shapeCode == null || shapeCode.isEmpty()) {
    errors.add("Missing or empty 'shape' parameter");
} else if(challengeIdAsString == null || challengeIdAsString.isEmpty()) {
    errors.add("Missing or empty challenge 'id'");
}
```

The SubmitCode servlet is used to submit a player's code for a given challenge and requires similar parameter checking, while the GameStateServlet sends the game state to the client device. In a similar fashion to SubmitCode, SubmitQuestionnaire sends data regarding a questionnaire completed by the player to the server for storage. For the administration side of aMazeChallenge, the AddChallengeServlet can be used to create a new challenge for online play while the AddParameterServlet and DeleteParameterServlet can be used by the administrator to create global parameters for the game. Perhaps the most important of all servlets is the RunEngineServlet, which cycles through each of the stored challenges and schedules it to be ran (see **Appendix 16**).

5.5 Summary

A combination of techniques and tools were used to carry out the implementation of aMazeChallenge. This stage was carried out in incremental and iterative steps and progress stagnated many times until a satisfiable solution was found. However, encountering these problems indirectly helped us ensure that a high-quality implementation was accomplished.

6 Test Strategy

6.1 Introduction

Perhaps the most challenging part of any project development cycle is testing. (Sullivan & Chillarege, 1991) identify three factors that affect the availability of software systems:

- High quality software (i.e. few defects shipped);
- Recovery mechanisms which mask any software errors that surface in the field;
- The capability for non-disruptive maintenance.

Designing and implementing perfect software is almost impossible. Even high-profile multi-billion-dollar companies¹⁵ or national organizations¹⁶ face enormous challenges in producing error-free solutions and usually scramble to patch their software not long after it has been released. With countless experiences in software failures, the process of testing has become an apparent inevitability. Among many other authors, (Khan, 2007) classifies the levels of testing as:

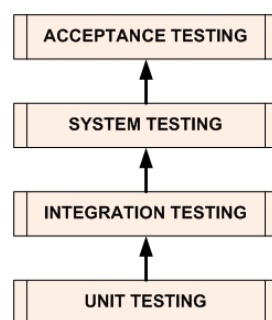


Figure 15 - Levels of testing

6.2 Unit Testing

The lowest level of testing deals with each individual unit of code. Several tools exist to help software engineers with testing at this level. For this project unit tests have been mainly implemented using JUnit, a Java unit testing framework for which Android Studio has built-in support and plug-ins. Two examples of the units of code that have been tested using this method are post-compilation code processing and questionnaire submissions. For the latter, a unit test can be found in **Appendix 17**.

¹⁵ Paypal (2007) - <https://it.slashdot.org/story/13/07/18/216247/paypal-credits-man-with-92-quadrillion>

¹⁶ Ariane 5 (1996) - https://en.wikipedia.org/wiki/Ariane_5

6.3 Integration testing

The purpose of integration testing is to reveal how well units of code work together and to reveal inconsistencies between these units as part of a group. To achieve this, functional requirements can be translated into tests. An integrated group of unit tests is expected to carry out a given set of functional requirements. As an example, the group of classes `InterpretedMazeSolver`, `InterpretedMazeSolverExecutor` and `InterpretedMazeSolverExecutorTask` are expected to carry out the task of interpreting the code written by the player. These classes need to be combined to achieve this specific objective and consequently had to be tested together as part of a larger JUnit test.

6.4 System testing

System testing is conducted on a completed and integrated system and attempts to evaluate its compliance with the entire set of requirements. At this stage it is important to assume different contexts in which the system might operate to make sure that it is compatible with a range of situations. A method to test a system is by using performance tests to measure the memory consumption and processing load required to execute a task. For example, a test was conducted related to the amount of time taken to compile a player's code. The table below shows data attained from such a test:

Galaxy J7				Galaxy Note 5				Emulator				Average
Average				Average				Average				
0.956	0.514	0.754	0.741333333	1.045	0.451	0.422	0.639333333	1.643	1.242	1.174	1.353	0.911222222

Table 2 - A table showing average times of compilation across different devices.

Another method used to test the system was regression testing. This kind of test ensured that the game continued to operate after the addition of new elements. Examples of new elements added to test the regression of the system include new pickable objects, new images, audio etc. In this kind of test, the system performed surprisingly well mostly due to its well-structured design. Important functions of the system have also been tested, including pickable object generation (their frequency and distribution), maze generation and the effects of pickable objects on the player. In some of these situations an ad hoc testing approach was employed to simulate a variety of random situations for each requirement, mostly while playing the game. Several bugs were revealed in the Blockly library (which is still in beta), the pickable objects' behaviour and the maze designer using this approach.

6.5 Acceptance testing

In the final stage of testing, the entire system is employed to achieve its intended objectives. The involvement of users in this stage can be of huge benefit because people with different mindsets tend to behave in different ways. At this stage, the user interface and its usability were evaluated and feedback was given by participants of an actual event on how to improve the application's interface to make it more usable and attractive. In general, the feedback provided was positive but some suggestions revealed different improvements that could be made.

6.6 Summary

The levels of testing suggested by (Khan, 2007) and the methods used address specific software failures outlined by (Sullivan & Chillarege, 1991) helped test aMazeChallenge from its smallest subcomponent to the entire system. A major part of acceptance testing is the evaluation of the system's achievements in terms of its intended objectives, which is what will be discussed next.

7 Evaluation, Conclusions and Future Work

7.1 Introduction

aMazeChallenge was tested and evaluated on March 8th at the University of Central Lancashire – Cyprus. The event organized by Dr Nearchos Paspallis involved a number of participants from a local high school that were introduced to programming through a brief lecture, were asked to play the game and complete a questionnaire regarding their experience. The project's evaluation is based on the data gathered during that event.

7.2 Project Objectives

1. Develop a game as an alternative method of teaching programming to students and engaging them in computing. The game's objectives and their attainment can be found in **Appendix 18**.
2. Measure how effective videogames can be for programming education and if they can provide a viable alternative to conventional teaching methods.
3. Introduce beginners to basic programming concepts and practices such as those described in section **1.2.2**.
4. Compare the skills and perception of programming of students with predisposition to computing (Code Cyprus 2017) with that of those without any prior experience or interest in it (Present study – high school group).

7.3 Evaluation

7.3.1 Player feedback

During the event a four-part questionnaire was handed to a group of high school students with an average age of 13.95 years. This questionnaire was mostly identical to another questionnaire answered by another group of students during the Code Cyprus 2017 event (Paspallis, et al., 2018). The first part of this questionnaire asked students how they rate their own skills in programming and mathematics and about their perception of programming, while the second part included some basic programming questions written in Blockly. These first two parts of the questionnaire were answered prior to playing the game and attempt to show the initial skills and perception of students. After playing the game, the group of a students was asked to complete the third and fourth parts. The third part consisted of questions similar to those answered in part two, while the fourth part contained more questions regarding the perception of programming by the students, similar to part one. In addition, several questions asked the students to rank which concept they did not understand the most and select some which feature of the game they enjoyed the most. The

objective of this approach is to expose the differences between the before and after results to questions regarding the student's opinion about programming and their skills in programming. Additionally, a comparison between the two questionnaires can be made to reveal differences between students who voluntarily enrolled to a programming event (prior disposition) and those who did not. While the high school group of students that took part in this experiment was not forced to participate, they were unaware of such an event taking place. This is in contrast to the group of students answering the same questions during Code Cyprus 2017, where everyone was voluntarily enrolled and had prior interest in programming and to all of the studied literature.

The number of participants during the event totalled to forty-three (43). The charts in **Appendix 19** reveal the distribution of gender of participants in both the high school visit and the Code Cyprus 2017 event. A difference can be seen between the genders of participants attending these events, with Code Cyprus being attended by 9% more women. A study conducted by (Roberts, et al., 2002) says that women are more likely to join a computer science course if the introductory modules are taught in an attractive and supportive way. Therefore, such events are believed to be a good way to engage women in computing.

The first two questions of the questionnaire regarded the self-rating of skills of students in mathematics and programming. The results in **Appendix 20** paradoxically show that the high school visit students rated their programming skills higher than those in mathematics while the students attending Code Cyprus ranked them lower. This is an unexpected outcome and may be a case of the Daniel-Kruger effect (Kruger & Dunning, 1999). The ranking of mathematics skills as higher by those in the high school visit could also imply that the general population is more interested in other physical sciences and math but not so much in computing, something that was illustrated in **Figure 1**.

The opinion of students about programming during the high school visit (**Appendix 21**) revealed that most students are excited or interested in programming with the most frequent choice being "interested". A significant amount of 16% was indifferent to programming before the event, which retained their opinion after playing the game. In addition, it is worth noting that the number of students who replied "excites" or "indifferent" before the game did not change, whereas those who replied "interests" (which is the majority) were lower by 26% after the game. Such a result may indicate that either the method used in this study was inappropriate to retain student interest, therefore contradicting the primary hypothesis that *games can be used as an alternative method to teach programming* or confirm the secondary hypothesis that *non-predisposed students will not be more interested in programming than those who are*. In direct contrast to these results, students attending Code Cyprus initially said that programming either excites them or interests them and this

amount was only reduced to 92% after attending the event. Despite the fact that Code Cyprus yields many advantages over the event carried out in this study, it is suspected that the secondary hypothesis is correct or at least partially correct as students from the high school visit had a significantly lesser opinion about programming.

The students' answers about the reasons for their interest in programming create a pattern between these results. **Appendix 22** shows that the students' responses were mixed, with 40% saying they "love programming", 23% saying they like programming because it can offer jobs while 35% were just "curious" about what it is. A significant amount of 21% selected the "Not interested" option while 16% said that they already knew programming. These results show that the majority of students liked programming before playing the game. Quite conversely, only 4% of the Code Cyprus students said that are not interested in programming while the overwhelming majority of 81% said that they love programming. These statistics confirm the gap that exists in programming disposition that was assumed at the start of this discussion and shows the real interest in programming of the general population.

An interesting part of this questionnaire is the difference between the participants' programming skills before and after playing the game. Participants answered several basic programming questions related to variables, loops and conditionals. A correct answer counted for one point and the number of correct answers was compared between the second and third part of the questionnaire. Some students answered questions in the second part of the questionnaire (before the game) but only a handful of students even attempted to complete similar questions in the third part. For those that did answer these questions, the difference in points shows a negative mark of -0.4 that indicates a decrease in programming knowledge. Whether that is the cause of external factors such as a short attention span or of the game itself is debatable. One reason for questioning these results is the fact that Code Cyprus students (who tend to be more receptive towards programming) also had a negative mark of -0.18 in the comparison of their answers before and after the game (**Appendix 23**). As a consequence of this the majority of students said that their skills were either unchanged or confused after the game, showing an awareness of their difficulties answering the questions. Students also answered questions regarding which concepts were most and least understood. The results shown in **Appendix 24** present that the concept of variables was both the most and least understood. The responses to these questions contradict each other and can only be interpreted by students not understanding either them or the concepts that were taught.

Although students had difficulties understanding the concepts taught and answering related questions, the majority of them (46%) said that they would likely or very likely follow a

career in programming (**Appendix 25**). This, shows that despite the difficulties they faced, participants were likely positively impacted by the game and would probably be more receptive to programming in the future. The answers to this question contradict the negative difference in opinions before and after the game. Inherently, they provide hope, and maybe proof, that *videogames can be used as an alternative method to teach programming* and that even *non-predisposed students can potentially become interested in programming* after taking part in such games.

Lastly, a question about the most enjoyed features of aMazeChallenge reveals that the most enjoyed feature was “graphics”, followed by “playing with others”. These results show what the students enjoyed most while playing the game and potentially indicate the most important aspects of the game that need to be improved in the future.

7.3.2 Objective attainment

7.3.2.1 Game features and performance

Overall, the project’s development part is regarded as a success. All of the features of the game prioritized as “must have”, “should have” and “could have” were successfully implemented and some of the “would have” objectives were also achieved. The system performs reasonably well as a whole, with good performance times, without any major bugs and is compliant with multiple devices running Android. Tests undertaken prior and during the high school visit event confirmed that the game is playable on a range of devices.

7.3.3 Difficulties faced

Worth noting is that several difficulties were faced during the event such as network connectivity problems. A network problem is a deal breaker for such events as aMazeChallenge’s multiplayer game requires an internet connection. In addition, external libraries such as Google’s Blockly and Mozilla’s Rhino created problems several times because of compatibility issues. These problems were not easily resolved and took away valuable time from the project’s implementation. Lastly, during the evaluation event, students naturally tended to have very short attention spans which may have had implications on their performance during the questionnaire.

7.3.4 Usefulness of solution and comparison with similar work

aMazeChallenge offers a useful tool that can be used to teach programming to beginners in a fun, interactive and immersive way. The results of this study can be used to confirm that games such as aMazeChallenge and those discussed by (Barnes, et al., 2007), (Harris & Jovanovic, 2010), (Chan, 2014), (Long, 2007) and many others can be used to create an environment where programming is not seen as a daunting and dull task but as a

creative, interactive, and fun experience that can allow someone to improve their skills by competing against or cooperating with other players. In contrast to the aforementioned literature, this study has also considered how the general population perceives programming and whether or not it is possible to convince a portion of sceptical students to study computing. Despite the mixed results, it is shown that it is possible to affect the opinions of these undetermined students. However, the usefulness of these results can be doubted. The impact of a single game on students is highly questionable and such questions can only be answered after other, larger groups of students have played a variety of educational programming games including aMazeChallenge. In addition, these results do not generalize for the entire world. It is commonly known that Cyprus lacks in terms of responsiveness in computing relative to other countries and has traditionally been a hub of other kinds of professions. In addition, despite most authors using questionnaires to estimate the success of their products, other ways of evaluating the success of videogames in programming education can also be used. Lastly, to answer if non-predisposed students can be convinced to study programming control groups similar to those used by (Barnes, et al., 2007) can be set up to compare differences in results before coming to conclusions.

7.4 Applicability of Findings to the Commercial World

The possibilities offered by the success of such games as teaching methods can have far reaching effects that are not necessarily limited to computing. Videogames can be utilized to teach any subject in the classroom in fun ways that do not affect the material being taught. Furthermore, aMazeChallenge and other games can be used during events to deliver crash courses in programming in an entertaining environment, therefore attracting young, undecided students to computing and indirectly increasing the number of computing graduates around the world. The gap between the demand and supply of software engineers is a problem that could be solved in part by the use of these entertaining educational games. The results are also applicable to experienced programmers. For example, games similar to aMazeChallenge can be used by companies to teach new programming languages and tools to their employees very quickly and during off-times. Both the competitive and cooperative nature of such games makes them ideal to use in a large set of contexts. Lastly, the adjective “educational” highlights the importance of such games as teaching tools and the potential of their contribution to a more technologically literate and well-educated society.

7.5 Conclusions

The aMazeChallenge project has resulted in the development of a game that can be used in a variety of contexts to teach basic programming concepts. The game is based on a mobile platform and therefore can be used by anyone, anywhere at any time, unlike most

other studies which have created games based on desktop computers or physical objects. aMazeChallenge is usable, maintainable and expandable and can provide a legitimate way to attract undecided students to computing. The study conducted has revealed insights on how these undecided students perceive programming and how the game has affected their opinions and skills. Despite the results being mixed, a conclusion that they positively affect the opinion of such students is made based on their willingness and likelihood to study programming in the future. Lastly, it confirms reports by other authors that games can provide a viable alternative to teach programming and in conjunction with other future researches can show that basic programming principles can be better learnt while playing a fun game.

7.6 Future Work

Despite a high completion rate among the game's required features, future work can implement tasks in **Table 1** that have not been completed. Because the game's code is easily expandable, new features could be introduced such as new pickable objects, different icon types for players, more avatars, a better point and ranking system etc. I personally believe that the introduction of animations will dramatically affect the realism and visual appeal of the game. In addition, a number of improvements in the user interface could be made to enhance the game's usability even further. In terms of networking, the application can consume a large amount of bandwidth because of the continuous processing of positions inside a maze. Fixes can be made to lower the bandwidth using specific software engineering techniques. In terms of the evaluation, a mix of results suggests that more experiments need to be conducted to establish a trend. The variety and number of participants and contexts need to be increased to provide more representative data samples. It is therefore implicit that more experiments need to be conducted in the future to confirm or deny the results of this study. Lastly, the game needs to be thoroughly tested using various other methods such as stress testing, load testing and simulations to improve its reliability and to establish its limitations in terms of participants, inputs and contexts.

7.7 Concluding Reflections

This project has been a milestone in my experience as a student, software engineer and scientist. In its duration, I have used my learning capabilities to full extent to devise solutions to problems never seen before. The accumulation of software engineering skills through the course has helped me design and implement a high-quality system. Most importantly, I have learnt to use critical thinking to analyse and compare existing literature, to find patterns in related work and to combine the insights provided by others with my own to reach conclusions to initial hypotheses.

8 References

1. AgileManifesto.org, 2009. *Twelve principles of agile software*. [Online]
Available at: <http://agilemanifesto.org/>
[Accessed 20 4 2018].
2. Babbie, E., 1990. *Survey Research Methods*. s.l.:Wadsworth Publishing Company.
3. Barnes, T. et al., 2007. *Game2Learn: A study of games as tools for learning*, s.l.: SIGCSE'07, USA.
4. Chan, S., 2014. *The educational effectiveness of a cooperative and competitive videogame for teaching introductory programming*, s.l.: McMaster University.
5. Clegg, D. & Barker, R., 1994. *Case method fast-track: A RAD approach*. 1st ed.
Boston: Addison-Wesley Longman publishing.
6. Combefis, S., Beresnevicius, G. & Dagiene, V., 2016. *Learning programming through games and contests: overview characterisation and discussion*, Vilnius: Olympiads in Informatics.
7. Eagle, M. & Barnes, T., n.d. *Experimental Evaluation of an Educational Game for Improved Learning in Introductory Computing*, Charlotte: University of North Carolina.
8. European Union, 1995. *EU Directive 95/46*. s.l.:s.n.
9. Harris, J. & Jovanovic, V., 2010. *Designing an introductory programming course using games*, s.l.: Georgia Southern University.
10. Hneif, M. & Ow, S., 2009. *Review of agile methodologies in software development*, Malaya: International Journal of Research and Review in Applied Science.
11. Jithin, 2016. *What is MVC? Advantages and Disadvantages*. [Online]
Available at: <https://www.interserver.net/tips/kb/mvc-advantages-disadvantages-mvc/>
[Accessed 3 4 2018].
12. Jobe, W., 2013. *Native Apps vs Mobile Web Apps*, Stockholm: Stockholm University.
13. Karch, M., 2018. *7 Free Programming Languages to teach kids how to code*. [Online]
Available at: <https://www.lifewire.com/kids-programming-languages-4125938>
[Accessed 10 4 2018].
14. Khan, E., 2007. *Different software testing levels for detecting errors*, Oman: Al Musanna College of Technology.
15. Kiss, G. & Arki, Z., 2016. *The influence of game-based programming education on the algorithmic thinking*, Almeria: 7th International Conference on Intercultural Education.
16. Klopfer, E., Osterweil, S. & Salen, K., 2009. *Moving learning games forward: obstacle, opportunities and openness*. Massachusetts: MIT.
17. Knutsson, B., Lu, H., Xu, W. & Hopkins, B., 2004. *Peer-to-Peer Support for massively multiplayer games*, Pennsylvania: IEEE.

18. Kruger, J. & Dunning, D., 1999. *Unskilled and unaware of it: how difficulties in recognizing one's own incompetence lead to inflated self-assessments*, Conell: American Psychological Association.
19. Laamarti, F., Eid, M. & Saddik, A., 2014. *An overview of serious games*, s.l.: Hindawi Publishing Corporation.
20. Leong, B., Koh, Z. H. & Razeen, A., 2011. *Teaching Introductory Programming as an Online Game*, Singapore: Department of Computer Science National University of Singapore.
21. Long, J., 2007. *Just for fun: using programming games in software programming training and education - a field study of IBM Robocode Community*, San Marcos, TX: Journal of Information Technology education.
22. Martin, T., Matthew, B., Benton, T. & Carmen, S., 2013. Learning programming with IPRO: The effects of a mobile, social gaming environment. *Journal of interacting learning research*, 24(3), pp. 301-328.
23. Mombrea, M., 2012. *When to use cloud platforms vs dedicated servers*. [Online] Available at: <https://www.itworld.com/article/2832631/cloud-computing/when-to-use-cloud-platforms-vs--dedicated-servers.html> [Accessed 11 4 2018].
24. Nielsen, J., 1995. *10 Usability Heuristics for User Interface Design*. [Online] Available at: <https://www.nngroup.com/articles/ten-usability-heuristics/> [Accessed 5 4 2018].
25. Paspallis, N. et al., 2018. *An experience report on the effectiveness of five themed workshops at inspiring high school students to learn coding*, Larnaca: ITICSE.
26. Pattis, R., 1981. *Karel the Robot: A Gentle Introduction to The Art of Programming*. 2nd ed. s.l.:Phoenix Color Corp.
27. Roberts, E., Kassianidou, M. & Irani, L., 2002. *Encouraging women in computer science*, Stanford: Stanford University.
28. Seng, W. & Wan, W., 2015. *The effectiveness of digital game for introductory programming concepts*, Selangor: ICITST.
29. Shih, W., 2017. *Mining learners' behavioral sequential patterns in a blockly visual programming educational game*, Taiwan: IEEE.
30. Soloway, E., 1986. *Learning to program = Learning to construct mechanisms and explanations*, s.l.: National Science Foundation.
31. Statista, 2017. *Global mobile OS market share in sales to end users from 1st quarter 2009 to 2nd quarter 2017*. [Online] Available at: <https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/> [Accessed 20 4 2018].
32. Sullivan, M. & Chillarege, R., 1991. *Software defects and their impact on system availability - a study of field failures in operating systems*, Yorktown Heights: IBM.

33. Sun Microsystems, 1997. *Java Code Conventions*. California: Sun Microsystems.
34. Vinicius, A. & James, D., 2016. *Learning programming patterns using games*, s.l.: IJICTE.
35. Weintrop, D. & Wilensky, U., 2016. *Playing by programming: making gameplay a programming activity*, s.l.: Education Technology.
36. Wintrop, D. & Wilensky, U., 2012. *RoboBuilder: A computational thinking game*, Evanston: Northwestern University.

9 Appendices

9.1 Appendix 1

Privacy Policy

The app requests access to the user account to simplify the collection of name/email in the Personalization screen. You can still set an alias name and email before connecting to the online server. Also, connecting to the online server (and thus registering to play with these credentials) is optional.

In some cases, the app will ask you to fill in a questionnaire. These data are collected purely for research purposes. The questionnaire does not include your email, but it does include a unique ID (per installation) which is used to identify which answers come from the same users, and to correlate answers with performance.

9.2 Appendix 2

Code designer and static checker states

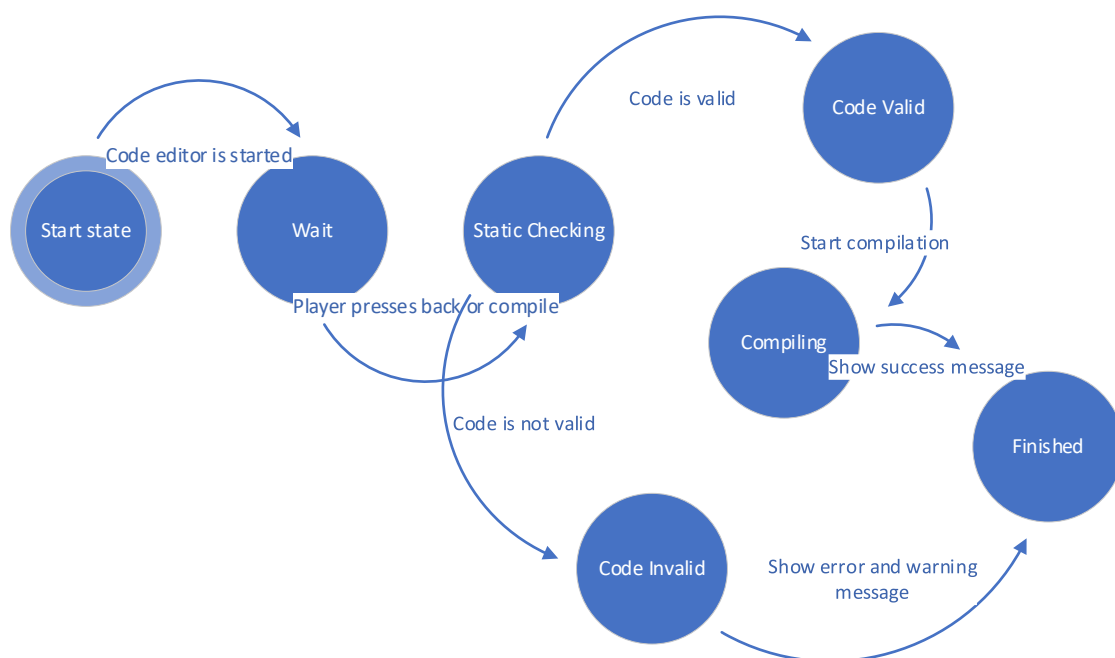


Figure 16 - Code designer and static checker state transition diagram

9.3 Appendix 3

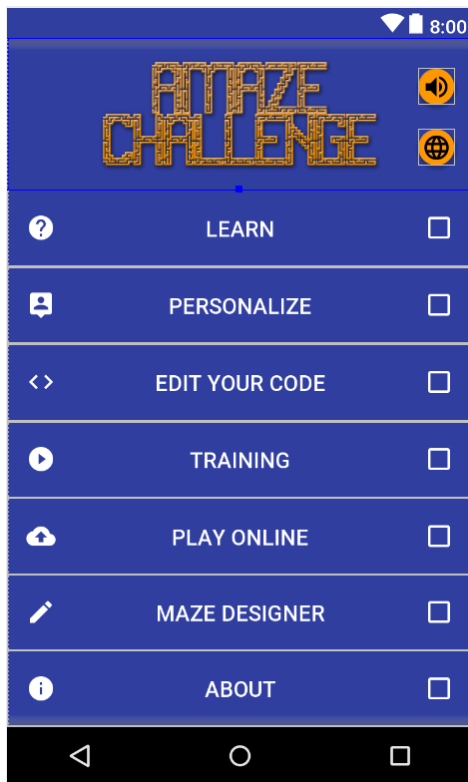


Figure 17 - Main Menu design

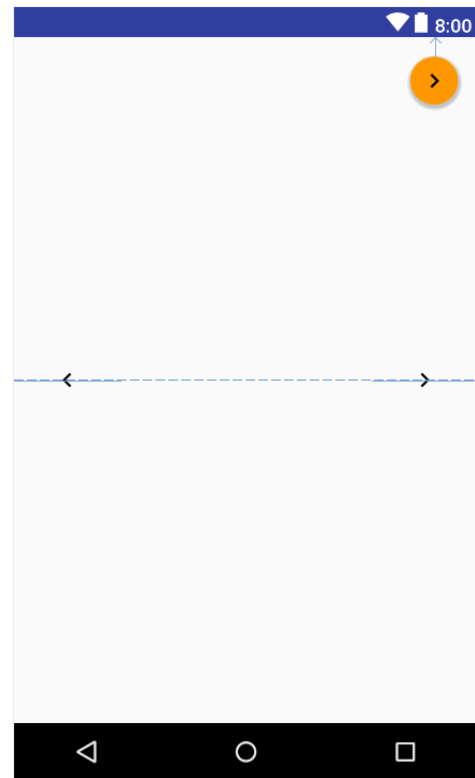


Figure 18 - Learning activity design

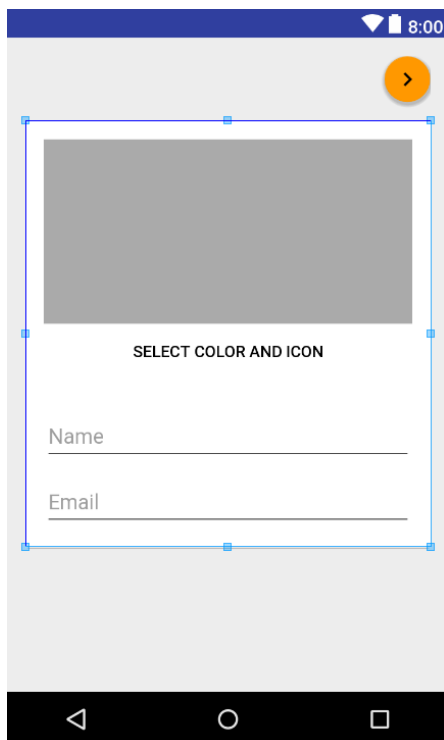


Figure 19 - Personalization activity design

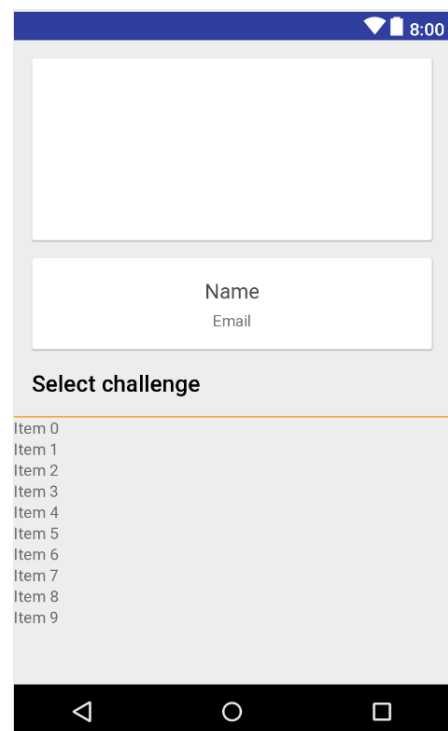


Figure 20 - Tutorial Activity design

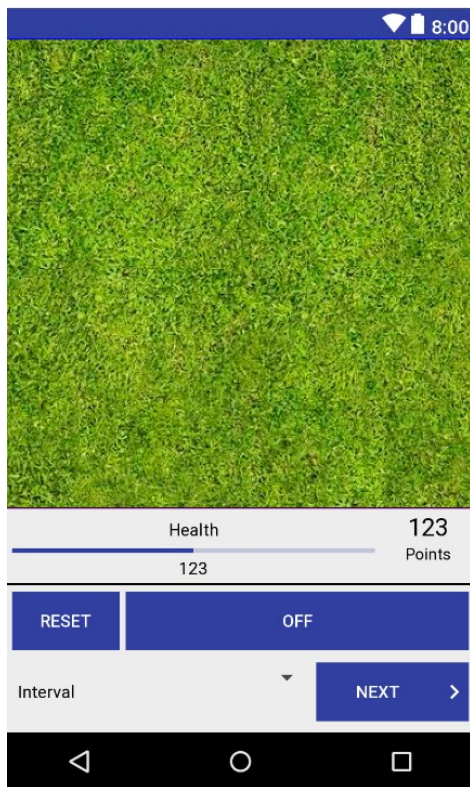


Figure 21 - Game Activity design

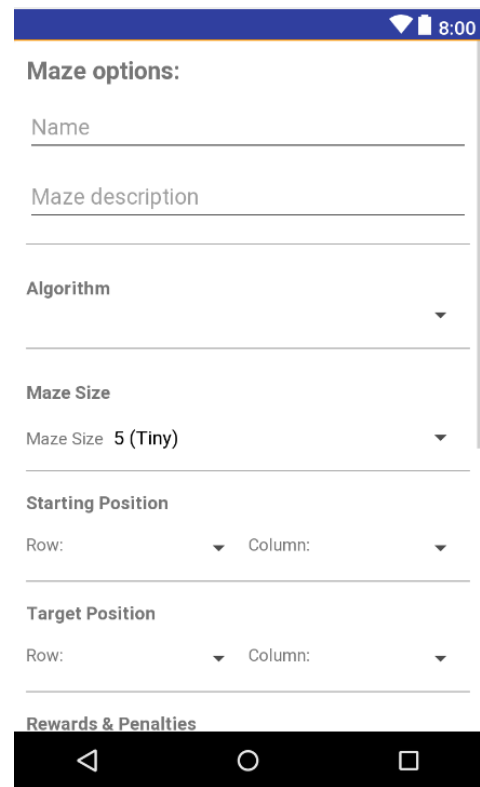


Figure 22 - Maze designer design

9.4 Appendix 4

void onDraw() - GameView.java

```
@Override
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);

    backgroundDrawable.setBounds(0, 0, canvas.getWidth(), canvas.getHeight());
    backgroundDrawable.setTileModeXY(Shader.TileMode.REPEAT, Shader.TileMode.REPEAT);
    backgroundDrawable.draw(canvas);

    if(grid == null) return;

    // compute tile_size and padding
    final int width = canvas.getWidth();
    final int height = canvas.getHeight();
    final int smallestSide = Math.min(width, height);
    // assumes the tiles are squares
    final int tile_size = smallestSide / grid.getWidth();
    final int padding = (smallestSide - (tile_size * grid.getWidth())) / 2;

    // draw maze grid (row 0 is top, and col 0 is left (so move from top left rightwards, then
    // next row, and so on)
    for(int row = 0; row < grid.getWidth(); row++) {
        for(int col = 0; col < grid.getHeight(); col++) {
            final int shape = RuntimeController.getGridCell(grid, row, col);
            drawGridCell(row, col, tile_size, padding, shape, lineColor, canvas);
        }
    }

    // draw starting and target positions
    final Position startingPosition = grid.getStartingPosition();
    drawGridCell(startingPosition.getRow(), startingPosition.getCol(), tile_size, padding,
0x0, COLOR_BLACK, COLOR_LIGHT_RED, canvas);
    final Position targetPosition = grid.getTargetPosition();
    drawGridCell(targetPosition.getRow(), targetPosition.getCol(), tile_size, padding, 0x0,
COLOR_BLACK, COLOR_LIGHT_GREEN, canvas);

    // draw pickables and rewards
    for(final Pickable pickable : pickables) {
        drawPickableItem(pickable.getPosition(), getDrawableResourceId(pickable), tile_size,
padding, canvas);
    }

    // draw active players
    for(final Map.Entry<String, PlayerPositionAndDirection> entry :
activePlayerIdToPositionAndDirectionMap.entrySet()) {
        final String activePlayerId = entry.getKey();
        final PlayerPositionAndDirection playerPositionAndDirection = entry.getValue();
        drawPlayer(allIdsToPlayers.get(activePlayerId),
playerPositionAndDirection.getPosition(), playerPositionAndDirection.getDirection(),
tile_size, padding, canvas);
    }
}
```

Listing 1 - onDraw() in GameView.java

9.5 Appendix 5

activity_blockly.xml (Device view)

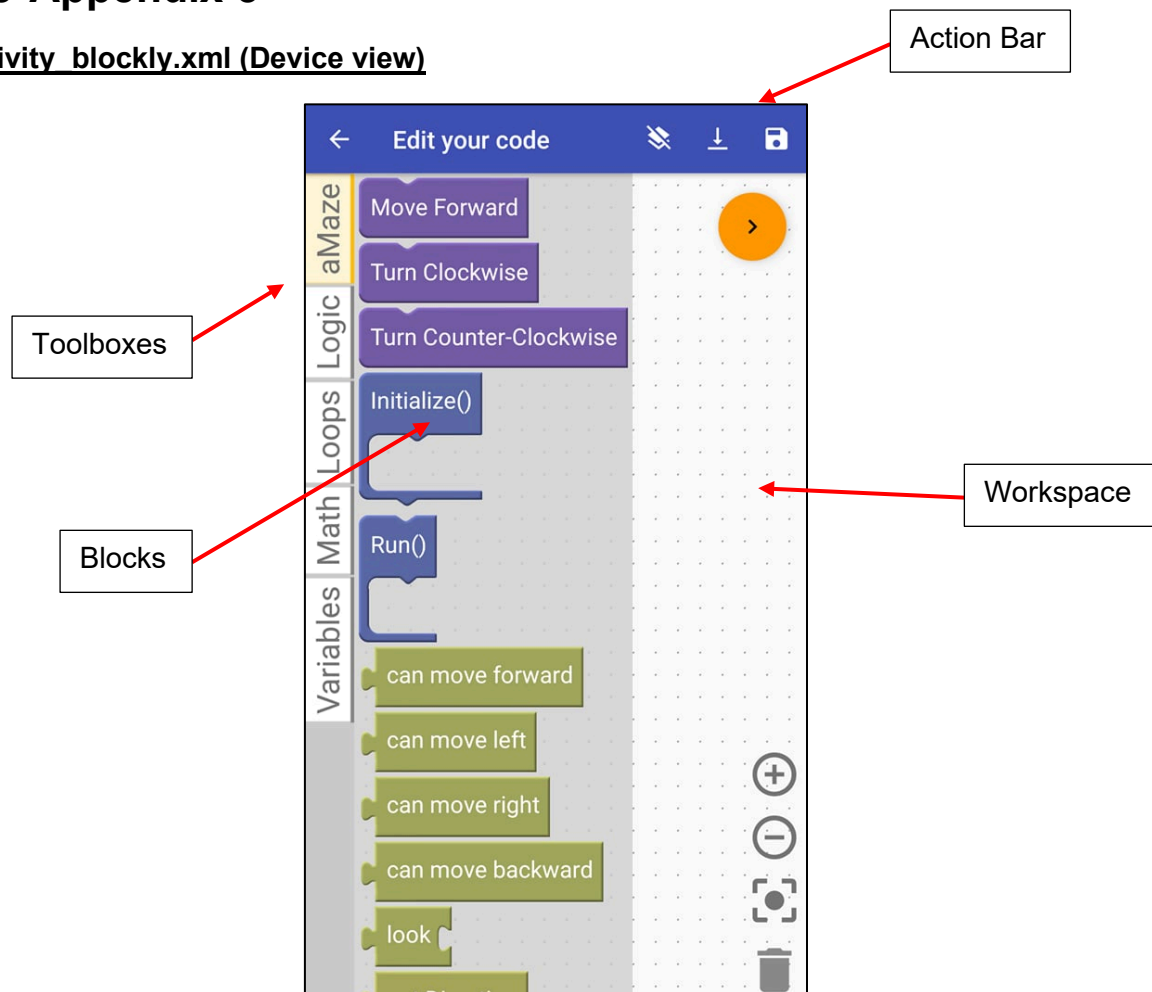


Figure 23 - Blockly code library

Note the customized toolbox created for aMazeChallenge. The customized toolbox was created using the Blockly Developer Tools. A link to these tools and the specific toolbox can be found below:

<https://blockly-demo.appspot.com/static/demos/blockfactory/index.html#64pqdg>

Blockly blocks are defined in XML files in assets/blocks and their generators are in assets/generators.

9.6 Appendix 6

Types of programming mistakes that are detectable:

1. Empty compound statements (such as loops without a body).
2. Empty conditional statements (such as if statements without a condition).
3. Empty Initialize or Run function.
4. Function in function statements (inserting a function statement into a function).
5. Infinite loops (obvious infinite loops such as while(true) are detectable).
6. Missing initialize or run function.
7. Run function redefinition.

Example code for a static checker (EmptyCompoundStatementFinder.java)

```
public class EmptyCompoundStatementFinder implements ErrorFinder {

    public static final InterpreterError ERROR = InterpreterError.EMPTY_STATEMENT_BODY;

    @Override
    public ArrayList<InterpreterError> execute(final String code) {
        ArrayList<InterpreterError> errorList = new ArrayList<>();
        final String[] lines = code.split("\n");
        for (int i = 0; i < lines.length; i++) {
            if (ErrorFinderCommons.lineContainsCompoundStatement(lines[i])) {
                boolean foundNextOrStatementDecl = false;
                for (int j = i; j < lines.length; j++) {
                    if (lines[j].contains(ErrorFinderCommons.XML_NEXT_STATEMENT_TAG)) {
                        errorList.add(ERROR);
                        foundNextOrStatementDecl = true;
                        break;
                    }
                    //end if line contains next decl
                }
                else if (lines[j].contains(ErrorFinderCommons.XML_STATEMENT_BODY)) {
                    foundNextOrStatementDecl = true;
                    break;
                }
                //end if line contains statement decl
            }
            //end foreach subsequent line
            if (!foundNextOrStatementDecl) errorList.add(ERROR);
        }
        //end if compound statement found
    }
    //end foreach line
    return errorList;
}
//end execute()

}
//end class EmptyCompoundStatementFinder
```

InterpreterError.java

```
public ArrayList<InterpreterError> executeErrorFinder(final String code) {
    Method requiredMethod = null;
    Method[] methods = errorFinder.getMethods();
    try {
        for (Method m : methods) {
            if (m.getName().equalsIgnoreCase(ErrorFinder.EXECUTE_METHOD_NAME)) {
                Class[] parameterClasses = m.getParameterTypes();
                requiredMethod = errorFinder.getMethod(ErrorFinder.EXECUTE_METHOD_NAME,
parameterClasses);
                break;
            }
        }
        //end if
    }
    //end for
    Object o = errorFinder.newInstance();
    return (ArrayList<InterpreterError>) requiredMethod.invoke(o, new Object[]{code});
}
//end try
catch (Exception e) { e.printStackTrace(); }
return null;
}
//end executeErrorFinder()
```

Listing 2 - Static checkers code

9.7 Appendix 7

JavaScript code injection – void processCode() in InterpreterMazeSolver.java:

```
private static String processCode(final String code) {
    final int declarationsEnd = getPlayerCodeStart(code);
    final String variableStoring = "\n\n\n/*Params Array*/\n" +
        "var propNames = [];\n" +
        "function populatePropNames() {\n"+
        "    for(var propName in this) {\n"+
        "        if (typeof this[propName] != 'function' && this[propName] != 'propName' &&
! (this[propName] instanceof Array)) {\n" +
        "            propNames.push(propName);\n" +
        "        }\n" +
        "    }\n" +
        "}\n" +
        "\n" +
        "function _getRandomInt(a, b) {\n" +
        "    if (a > b) {\n" +
        "        var c = a;\n" +
        "        a = b;\n" +
        "        b = c;\n" +
        "    }\n" +
        "    return Math.floor(Math.random() * (b - a + 1) + a);\n" +
        "}\n";
    String startingCode = "";
    if (declarationsEnd > 0) startingCode = code.substring(0, declarationsEnd);
    String endingCode = "";
    if (declarationsEnd > 0) endingCode = code.substring(declarationsEnd, code.length());
    else endingCode = code;
    final String finalCode = startingCode + variableStoring + endingCode;
    return filterCode(finalCode);
}
```

Listing 3 - processCode() in InterpreterMazeSolver.java

9.8 Appendix 8

Running the code

InterpretedMazeSolver.java

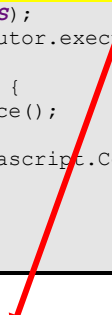
```
@Override
public PlayerMove getNextMove(final Game game) {
    this.game = game;

    PlayerMove nextMove = PlayerMove.NO_MOVE;

    try {
        final org.mozilla.javascript.Context rhinoContext =
org.mozilla.javascript.Context.enter();
        rhinoContext.setOptimizationLevel(-1);
        final org.mozilla.javascript.ScriptableObject scope =
rhinoContext.initStandardObjects();
        rhinoContext.evaluateString(scope, code, RUN_FUNCTION, 1, null);
        final org.mozilla.javascript.Function function = (org.mozilla.javascript.Function)
scope.get(RUN_FUNCTION, scope);

        InterpretedMazeSolverExecutor executor = new InterpretedMazeSolverExecutor(function,
this, TIME_LIMIT_SECONDS);
        nextMove = executor.execute();
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    finally {
        org.mozilla.javascript.Context.exit();
    }

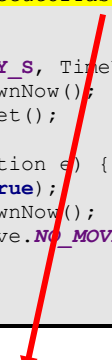
    return nextMove;
}
```



InterpretedMazeSolverExecutor.java

```
PlayerMove execute() throws Exception {
    ExecutorService executor = Executors.newSingleThreadExecutor();
    Future<PlayerMove> future = executor.submit(new
InterpretedMazeSolverExecutorTask(FUNCTION, INSTANCE));

    try {
        future.get(DELAY_S, TimeUnit.SECONDS);
        executor.shutdownNow();
        return future.get();
    }
    catch (TimeoutException e) {
        future.cancel(true);
        executor.shutdownNow();
        return PlayerMove.NO_MOVE;
    }
}
```



InterpretedMazeSolverExecutorTask.java

```
@Override
public PlayerMove call() throws Exception {
    Object result = FUNCTION.call(Context.enter(), Context.enter().initStandardObjects(),
Context.enter().initStandardObjects(), new Object[] { INSTANCE });
    return (PlayerMove) Context.jsToJava(result, PlayerMove.class);
} //end call()
```

Listing 4 - Code interpreting the player's code.

9.9 Appendix 9

Left wall follower in Blockly:

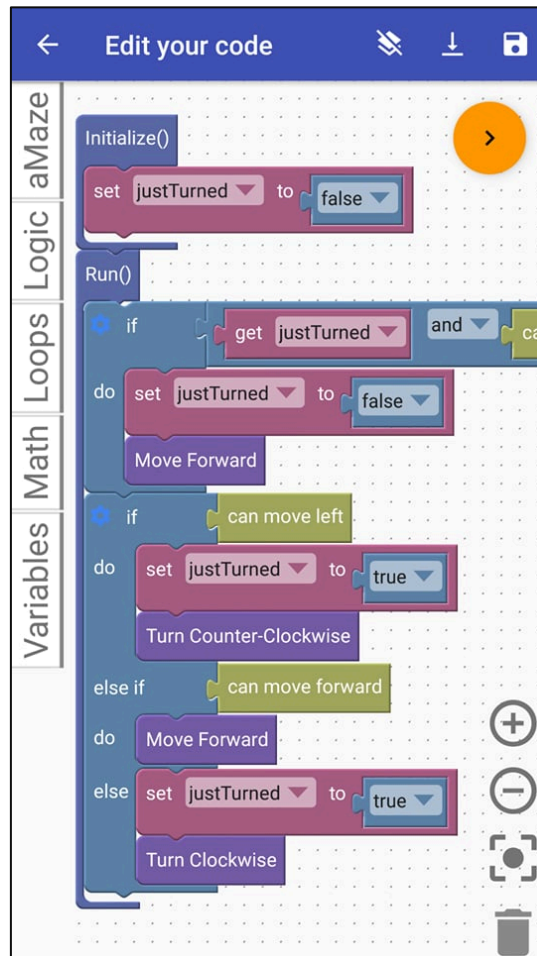


Figure 24 - An example implementation of the left wall follower algorithm in Blockly

9.10 Appendix 10

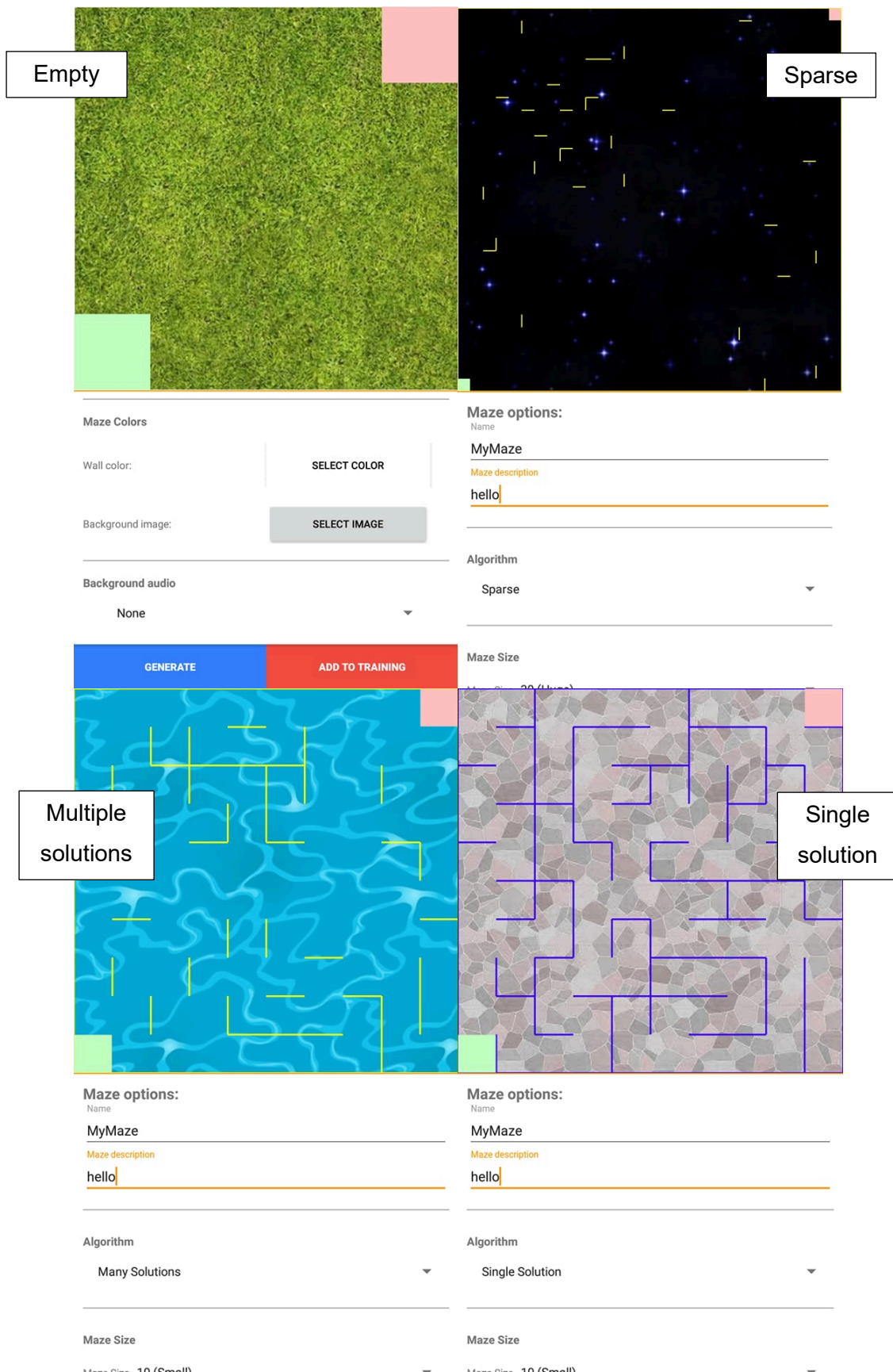


Figure 25 - A set of screenshots demonstrating the generation of different types of mazes.

9.11 Appendix 11

Pickable object behaviour:

RuntimeController.java

handlePickableState()

```
public static void handlePickableState(Game game) {
    for (int i = 0; i < game.getPickables().size(); i++) {
        game.getPickables().get(i).reduceState();
        if (game.getPickables().get(i).getState() <= 0) game.removePickupItem(i);
    }
}
```

Listing 5 – handlePickableState() in RuntimeController.java

applyPlayerMove()

```
if (pickable.getPosition().equals(position)) {

    //Health-related pickables:
    if (pickable.getPickableType() == PickableType.BOMB) {
        if (pickable.getState() == 1 || pickable.getState() == 2)

game.getPlayerById(playerId).getHealth().changeBy(pickable.getPickableType().getHealthChange());
    }
    else
game.getPlayerById(playerId).getHealth().changeBy(pickable.getPickableType().getHealthChange());

    //Apply effects of point-related pickables:
    game.getPlayerById(playerId).changePointsBy(pickable.getPickableType().getPointsChange());

    //Apply effects of speed-related pickables:
    if (pickable.getPickableType() == PickableType.SPEEDHACK) {
        addPlayerDoubleTurnsById(playerId, PickableType.SPEEDHACK_TURNS_AMOUNT);
    } else if (pickable.getPickableType() == PickableType.TRAP) {
        addPlayerLostTurnsById(playerId, PickableType.TRAP_TURNS_AMOUNT);
    }

    // if audio event listener set, notify with event
    if (audioEventListener != null && pickable.getPickableType() != PickableType.BOMB)
audioEventListener.onAudioEvent(pickable);
    if (pickable.getPickableType() != PickableType.BOMB) game.removePickupItem(i);
}
```

Listing 6 - applyPlayerMove() in RuntimeController.java

See next page for a screenshot of a game containing pickable objects.

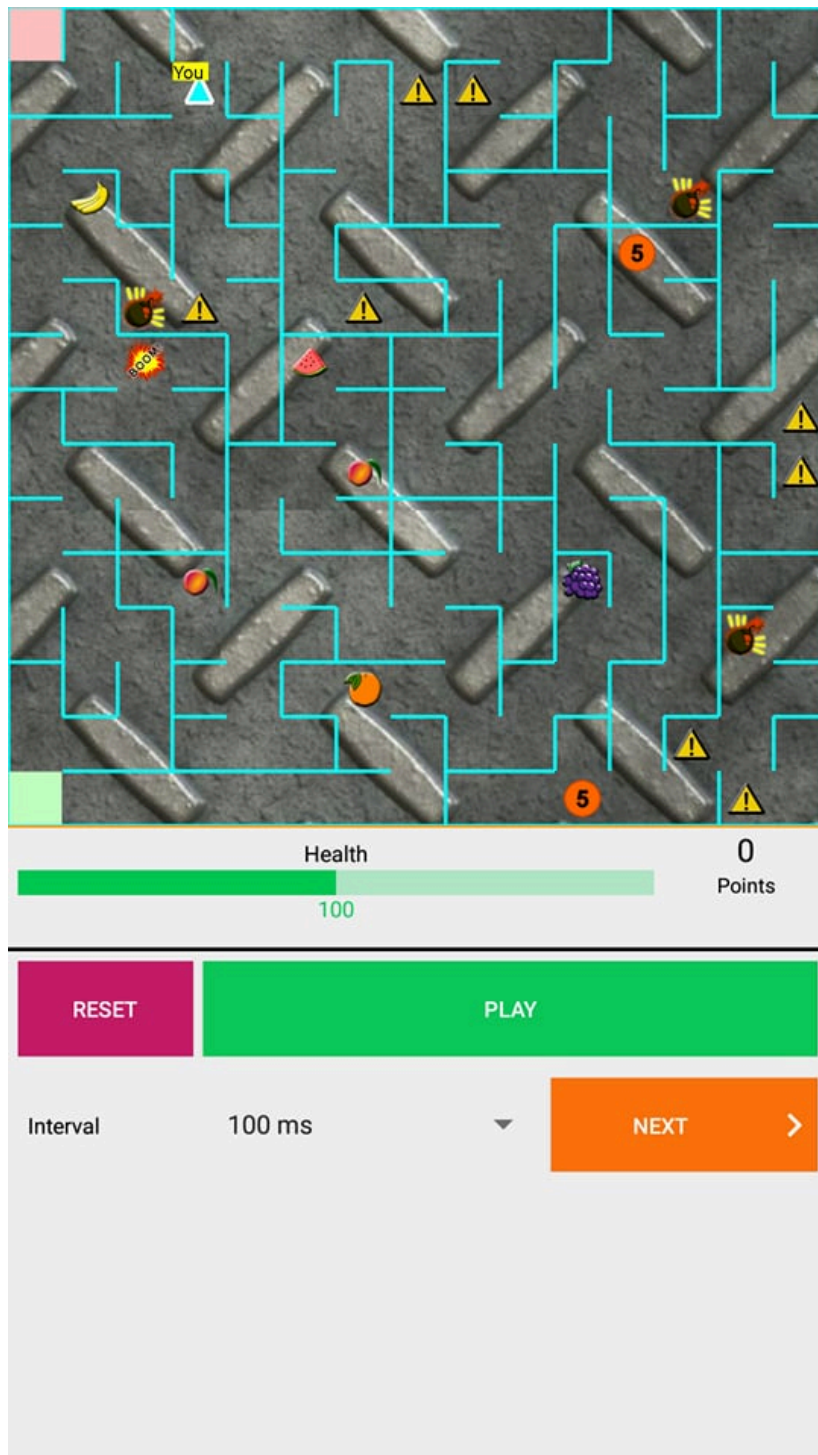


Figure 26 - A screenshot showing the generation and drawing of pickable objects.

See next page for a sample representation of a maze challenge in JSON format.

```

{
  "id": 7,
  "apiVersion": 1,
  "name": "8 - Park Adventure",
  "description": "A walk in the park!",
  "difficulty": "EASY",
  "createdOn": 0,
  "createdBy": "admin",
  "canRepeat": true,
  "canJoinAfterStart": true,
  "canStepOnEachOther": true,
  "minActivePlayers": 1,
  "maxActivePlayers": 10,
  "startTimestamp": 0,
  "endTimestamp": 0,
  "hasQuestionnaire": false,
  "rewards": "LOW",
  "penalties": "LOW",
  "algorithm": "SINGLE_SOLUTION",
  "grid": {
    "width": 10,
    "height": 10,
    "data":
"5333333395b6953395acd78695a69cc523a69d6acedd5b6295a522853128dc596a587ac684b5a4bdc7a69e509687332bee63a",
    "startingPosition": {
      "row": 9,
      "col": 0
    },
    "targetPosition": {
      "row": 0,
      "col": 9
    }
  },
  "lineColor": "#FFFF00",
  "backgroundImage": "TEXTURE_GRASS",
  "backgroundAudio": "AMBIENT_TROPICAL_FOREST"
}

```

9.12 Appendix 12

Personalization options:

PersonalizationActivity.java

```
private EditText nameEditText;
private EditText emailEditText;
private GIFView gifView;
private Button selectColorButton;

...

// load personalization preferences
final SharedPreferences sharedPreferences =
PreferenceManager.getDefaultSharedPreferences(this);
// update user color
{
    final String userColorName = sharedPreferences.getString(PREFERENCE_KEY_COLOR,
AmazeColor.BLACK.name());
    final AmazeColor userAmazeColor = AmazeColor.valueOf(userColorName);
    final int userColor = Color.parseColor(userAmazeColor.getHexCode());
    selectColorButton.setBackgroundColor(userColor);

selectColorButton.setTextColor(isBrightColor(Color.parseColor(userAmazeColor.getHexCode())) ?
Color.BLACK : Color.WHITE);
}
// update user icon/avatar
{
    final String userIconName = sharedPreferences.getString(PREFERENCE_KEY_ICON,
AmazeIcon.ICON_1.getName());
    final AmazeIcon selectedAmazeIcon = AmazeIcon.getByName(userIconName);
    gifView.setImageResource(getDrawableResourceId(selectedAmazeIcon));
}
// update user email and name
{
    final String email =
PreferenceManager.getDefaultSharedPreferences(this).getString(PREFERENCE_KEY_EMAIL, "");
    final String name =
PreferenceManager.getDefaultSharedPreferences(this).getString(PREFERENCE_KEY_NAME,
getString(R.string.Guest));
    if(email.isEmpty()) {
        if (checkSelfPermission(Manifest.permission.GET_ACCOUNTS) !=
PackageManager.PERMISSION_GRANTED) {
            requestPermissions(new String[] { Manifest.permission.GET_ACCOUNTS },
PERMISSIONS_REQUEST_GET_ACCOUNT);
        } else {
            selectAccount();
        }
    } else {
        emailEditText.setText(email);
        nameEditText.setText(name);
    }
}
```

Listing 7 – Code showing the use of preferences to save personalization data.

PersonalizationSliderActivity.java

```
private ViewPager.OnPageChangeListener iconOnPageChangeListener;
private ViewPager.OnPageChangeListener colorOnPageChangeListener;

...

//Icon
iconPagerAdapter = new IconPagerAdapter(getSupportFragmentManager());
iconPager.setAdapter(iconPagerAdapter);
iconPager.setPageTransformer(true, new ZoomOutPageTransformer());
previousIcon = findViewById(R.id.avatar_previous);
nextIcon = findViewById(R.id.avatar_next);
```

```

//Color
colorPagerAdapter = new ColorPagerAdapter(getSupportFragmentManager());
colorPager.setAdapter(colorPagerAdapter);
colorPager.setPageTransformer(true, new ZoomOutPageTransformer());
previousColor = findViewById(R.id.color_previous);
nextColor = findViewById(R.id.color_next);

...

iconOnPageChangeListener = new ViewPager.OnPageChangeListener() {
    @Override public void onPageScrolled(int position, float positionOffset, int
positionOffsetPixels) { /* nothing */ }

    @Override
    public void onPageSelected(int position) {
        newIcon = AmazeIcon.values()[position];
    } //end onPageSelected()

    @Override
    public void onPageScrollStateChanged(int state) { }
};

colorOnPageChangeListener = new ViewPager.OnPageChangeListener() {
    @Override
    public void onPageScrolled(int position, float positionOffset, int positionOffsetPixels) { }

    @Override
    public void onPageSelected(int position) { newColor = AmazeColor.values()[position]; }

    @Override
    public void onPageScrollStateChanged(int state) { }
};

```

Listing 8 - Code demonstrating the use of ViewPagers to select icons and colors.

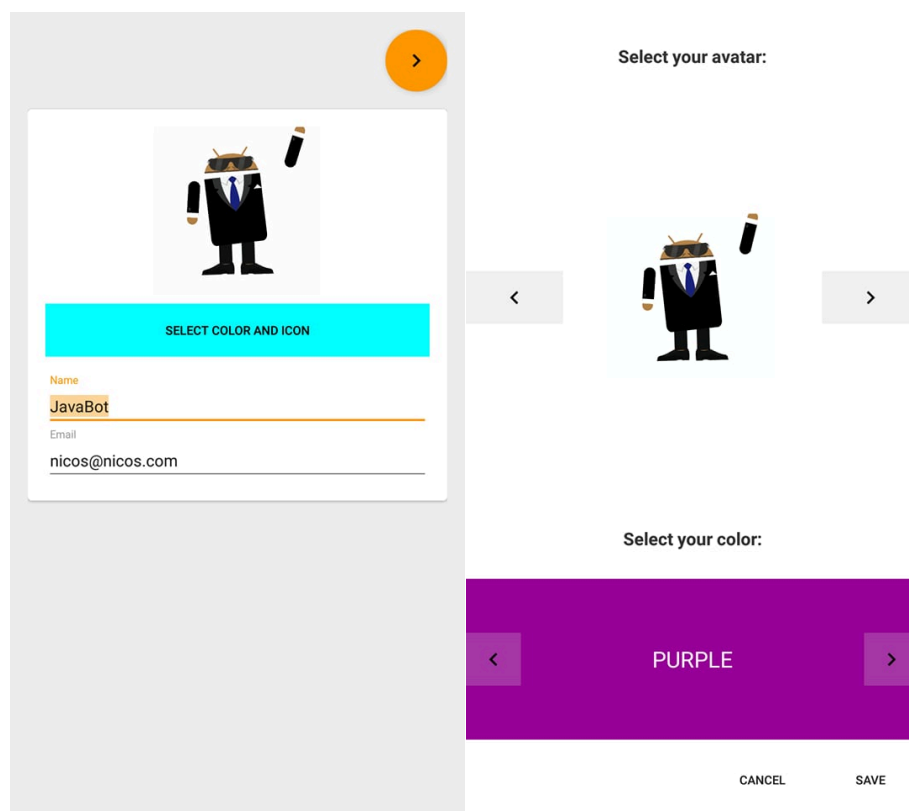


Figure 27 - Screenshots showing the implemented personalization activities and ViewPagers.

9.13 Appendix 13

Found in assets/help



Figure 28 - Screenshots showing several help/learning pages creating using HTML/CSS.

9.14 Appendix 14

Questionnaire screenshot

Questionnaire

Questionnaire Instructions:

Please answer questions honestly. Do not provide any more information than what is required. If you are not sure what a question is asking, please speak to someone who can help. You may choose to skip this questionnaire but your feedback would be highly appreciated.

SKIP

How was your experience playing aMaze Challenge?

★ ★ ★ ★ ★

Terrible OK Awesome

Would you share this experience with your friends?

☐ No ☐ Maybe ☐ Yes

How much do you believe this experience helped you learn more about programming?

Figure 29 - A screenshot showing the implemented questionnaire activity.

```
public enum LikertResponse {  
  
    VERY_POSITIVE("Very Positive"),  
    POSITIVE("Positive"),  
    NEUTRAL("Neutral"),  
    NEGATIVE("Negative"),  
    VERY_NEGATIVE("Very Negative")  
  
    ;  
  
    private String name;  
  
    LikertResponse(String name) {  
        this.name = name;  
    }  
  
    @Override  
    public String toString() {  
        return name;  
    }  
}  
  
public enum MultipleChoiceResponse {  
  
    CHOICE_1("Choice 1"),  
    CHOICE_2("Choice 2"),  
    CHOICE_3("Choice 3"),  
    CHOICE_4("Choice 4")  
  
    ;  
  
    private String name;  
  
    MultipleChoiceResponse(String name) {  
        this.name = name;  
    }  
  
    @Override  
    public String toString() {  
        return name;  
    }  
}
```

Listing 9 - Code showing how enumerating classes are used to represent types of responses for the questionnaire.

9.15 Appendix 15

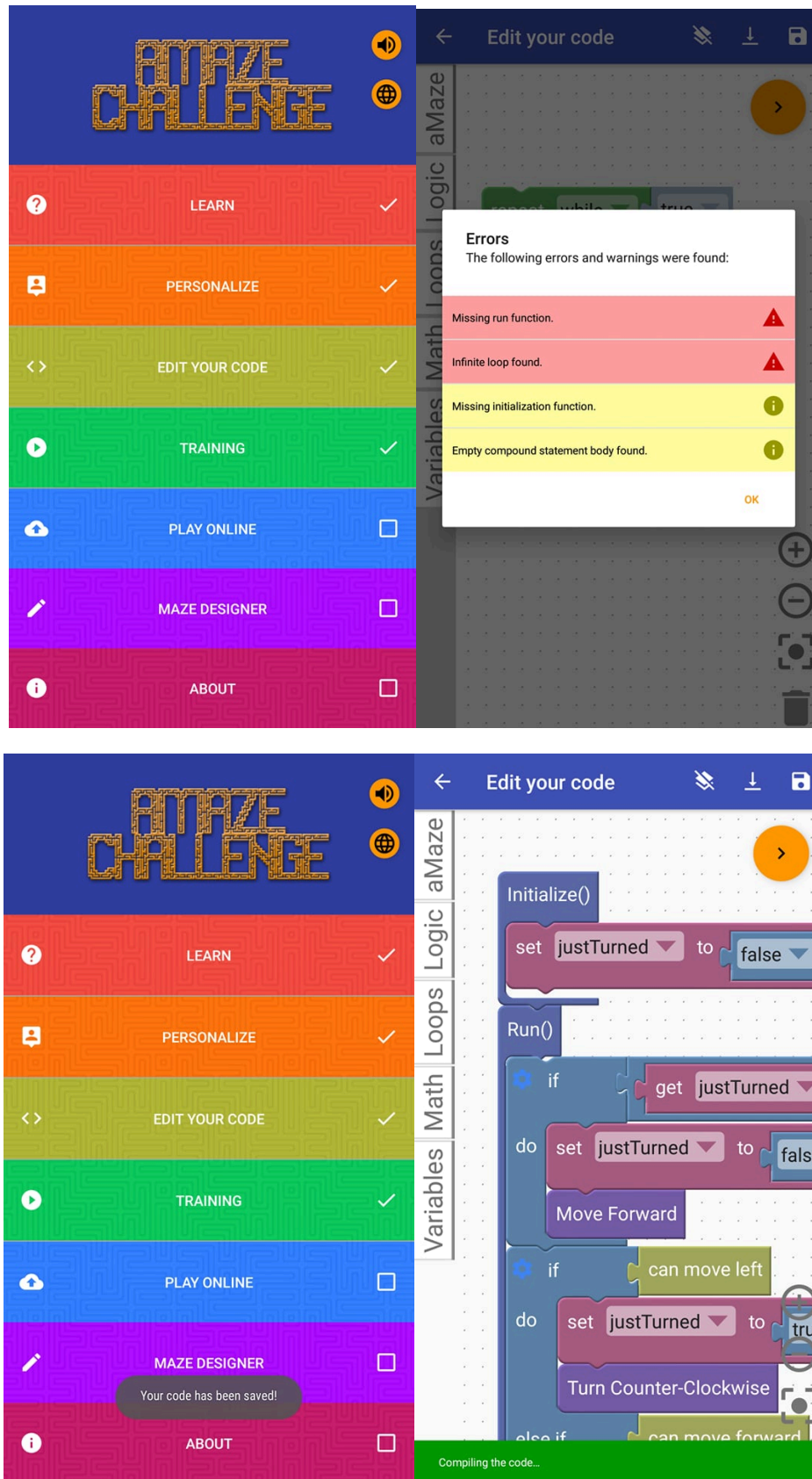


Figure 30 - Screenshots showing the implementation of the user interface.

9.16 Appendix 16

RunEngineServlet.java

```
if(challenge.isActive()) {
    final Queue queue = QueueFactory.getDefaultQueue();
    TaskOptions taskOptions = TaskOptions.Builder
        .withUrl("/admin/run-engine")
        .param("magic", magic)
        .param("challenge", Long.toString(challengeId))
        .param("game", Long.toString(gameId))
        .countdownMillis(ONE_SECOND)
        .method(TaskOptions.Method.GET);
    queue.add(taskOptions);
}
```

Listing 10 - Code showing how the server queues an instruction for the game's engine to run.

```
private Game implementGameLogic(final Challenge challenge, final Game game) {
    final long startTime = System.currentTimeMillis();

    final Grid grid = challenge.getGrid();

    // check if we can upgrade any players from 'waiting' to 'queued'
    final List<String> waitingPlayerIDs = game.getWaitingPlayerIDs();
    for(final String waitingPlayerId : waitingPlayerIDs) {
        final String code = (String)
memcacheService.get(SubmitCodeServlet.getMazeCodeKey(challenge.getId(), waitingPlayerId));
        if(code != null) {
            game.queuePlayerById(waitingPlayerId);
        }
    }

    // now check if we can upgrade any players from 'queued' to 'active'
    while(game.getNumberOfActivePlayers() < challenge.getMaxActivePlayers() &&
game.hasQueuedPlayers()) {
        // activate players as needed
        game.activateNextPlayer(grid);
    }

    // prepare active players
    final Map<String, MazeSolver> playerIDsToMazeSolvers = new HashMap<>();
    final List<String> activePlayerIDs = game.getActivePlayerIDs();
    for(final String activePlayerId : activePlayerIDs) {
        final String code = (String)
memcacheService.get(SubmitCodeServlet.getMazeCodeKey(challenge.getId(), activePlayerId));
        final MazeSolver mazeSolver = new InterpretedMazeSolver(challenge, game,
activePlayerId, code);
        mazeSolver.init(challenge, game);
        final byte [] state = (byte[]) memcacheService.get(getMazeSolverStateKey(game.getId(),
activePlayerId));
        mazeSolver.setState(state);
        playerIDsToMazeSolvers.put(activePlayerId, mazeSolver);
    }

    // todo ... consider revising to make multi-threaded and with deadlines? [e.g. check out:
com.google.appengine.api.ThreadManager]
    RuntimeController.makeMove(challenge, game, playerIDsToMazeSolvers);

    // store maze solvers' state to memcache
    for(final String activePlayerId : activePlayerIDs) {
        final MazeSolver mazeSolver = playerIDsToMazeSolvers.get(activePlayerId);
        memcacheService.put(getMazeSolverStateKey(game.getId(), activePlayerId),
mazeSolver.getState());
    }

    // remove completed players (move from 'active' to 'finished')
    final Position targetPosition = challenge.getGrid().getTargetPosition();
    for(final String activePlayerId : activePlayerIDs) {
        final Position playerPosition = game.getPositionById(activePlayerId);
        // for any players that were moved in 'inactive' status, reset their state and code so
they are not restarted automatically
    }
}
```

```

        if(playerPosition != null && playerPosition.equals(targetPosition)) {
            memcacheService.delete(getMazeSolverStateKey(game.getId(), activePlayerId)); //
            reset algorithm's state
            memcacheService.delete(SubmitCodeServlet.getMazeCodeKey(challenge.getId(),
            activePlayerId)); // reset submitted code
            game.resetPlayerById(activePlayerId);
        }

        // update game with number of rounds executed
        game.touch(System.currentTimeMillis() - startTime);

        return game;
    }

```

Listing 11 - Code showing how the server handles players by organizing them into queues for games.

9.17 Appendix 17

Unit test for questionnaire submission:

```
@Test
public void useAppContext() throws Exception {
    //Context of the app under test.
    Context appContext = InstrumentationRegistry.getTargetContext();

    final QuestionEntry [] questionEntries = new QuestionEntry [] {
        new QuestionEntry("hello", "world"),
        new QuestionEntry("hello2", "world2")
    };
    final QuestionnaireEntry questionnaireEntry = new
QuestionnaireEntry(Installation.id(appContext), 0L, questionEntries);
    final String json = new
GsonBuilder().setPrettyPrinting().create().toJson(questionnaireEntry);
    System.out.println("json: " + json);
    System.out.println("SubmitQuestionnaireAsyncTask...");
    new QuestionnaireActivity.SubmitQuestionnaireAsyncTask(appContext, json).execute();
    System.out.println("sleep...");
    try { Thread.currentThread().sleep(1000); } catch (InterruptedException ie) {}
    System.out.println("woke up!");
    assertEquals("", "");
}
```

Listing 12 - Code showing how a JUnit test is used to test the questionnaire submission.

9.18 Appendix 18

Game feature related objectives (Green = completed, Red = not completed):

	Objective	Priority	Attainment
1	Implemented on an appropriate cloud-based platform such as AppEngine.	MUST HAVE	AppEngine based server-side implementation
2	Provides a view which visualizes the game progress and players' standing in real time, implemented on an appropriate platform.	MUST HAVE	A view is provided in the Android app, in OnlineGameActivity
3	Provides a mobile view which enables the players to register to the game, define and test their code, and view their progress, on an appropriate platform for mobile devices such as the Web or Android	MUST HAVE	Such views are provided in Android in BlocklyActivity and GameActivity.
4	Realizes appropriate game elements, including graphics and sounds which enhance the game experience.	MUST HAVE	2D Graphics and game audio.
5	Includes interactive elements such as avatars, icons with colors that represent the player and can be changed in a personalization screen.	MUST HAVE	Includes avatars, colors for player icons and a personalization view under PersonalizationActivity
6	Incorporates a language that defines how the player will move and interact with objects inside the maze.	MUST HAVE	Utilizes Blockly Language
7	Allows a player to learn about the game, its rules and how it is played using a training scenario and help pages.	SHOULD HAVE	Players can learn about the game in HelpActivity and can practice by selecting mazes in TrainingActivity.
8	Utilizes a language that is highly usable for mobile devices and appropriate for beginners.	SHOULD HAVE	The Blockly language is a beginner-friendly language that is highly usable on mobile devices.
9	Uses a built-in mechanism that checks the player's code for obvious errors.	SHOULD HAVE	A static check has been implemented (ErrorFinder)
10	Include basic preset algorithms that can be used by players to compete against or changed to write their own code.	SHOULD HAVE	Basic algorithms such as left wall follower have been implemented and can be loaded and edited by players.
11	Include features that promote the game experience such as different objects that affect the player's health or points (e.g time bombs, traps, fruits and more)	SHOULD HAVE	"Pickable" objects have been implemented in the game that affect the player's health and points.

12	Allows the player to control the sound and vibration by toggling them on or off.	COULD HAVE	The player can control the sound and vibration through a button on the main screen (MainActivity)
13	Features multiple languages and allows the player to change between them.	COULD HAVE	The player can change between English and Greek through a button on the main menu (MainActivity)
14	Features a maze designer that allows players to design and play their own mazes.	WOULD HAVE	A maze designer and generator are featured which allow players to design their own mazes with a selection of predefined images, audio, wall colors and more. This can be done in MazeDesignerActivity.
15	Includes video tutorials that explain how to play and write code.	WOULD HAVE	Videos have not been included in this release of the game due to time limitations.
16	Uses 3D graphics and more advanced game elements such as animations to enhance the game experience	WOULD HAVE	3D Graphics, animations and other enhancements have not been implemented due to time limitations

Table 3 - A table showing objective attainment for aMazeChallenge.

9.19 Appendix 19

Gender distribution during high school visit

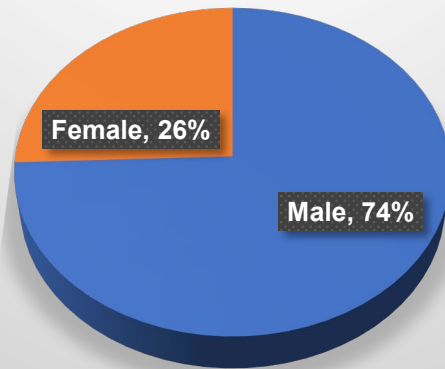


Figure 31 - A pie chart showing the distribution of gender during the high school visit

Gender distribution during Code Cyprus 2017

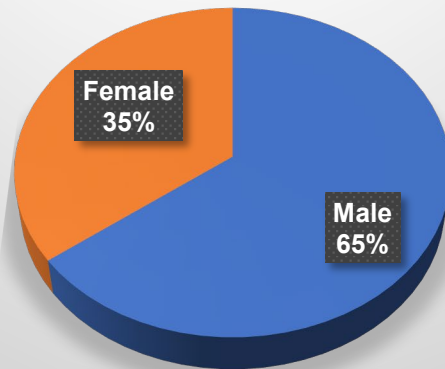
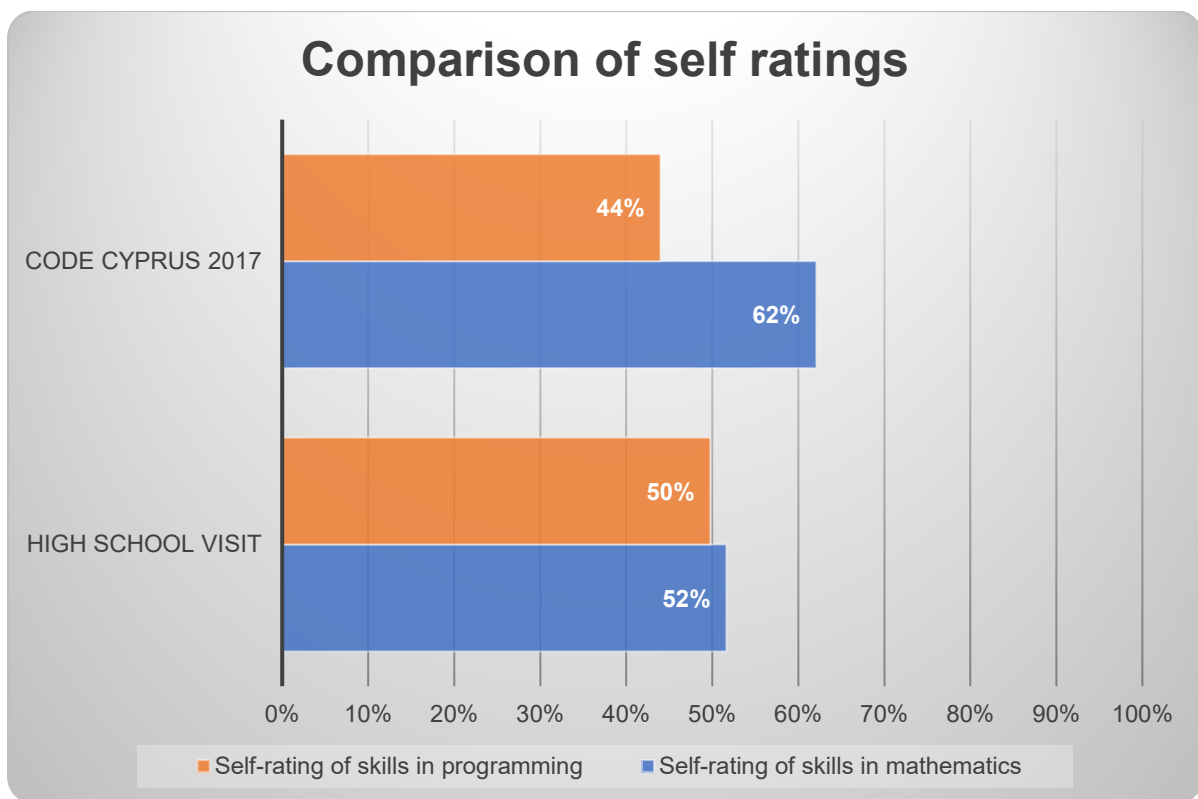


Figure 32 - A pie chart showing the distribution of gender during Code Cyprus 2017

9.20 Appendix 20



Note: the data has been adjusted to show the average rating of skill rated by the students on a 0-100% scale with 0% being not competent at all and 100% being extremely competent.

Figure 33 - A bar chart showing self-rating of students related to mathematics and programming during both events.

9.21 Appendix 21

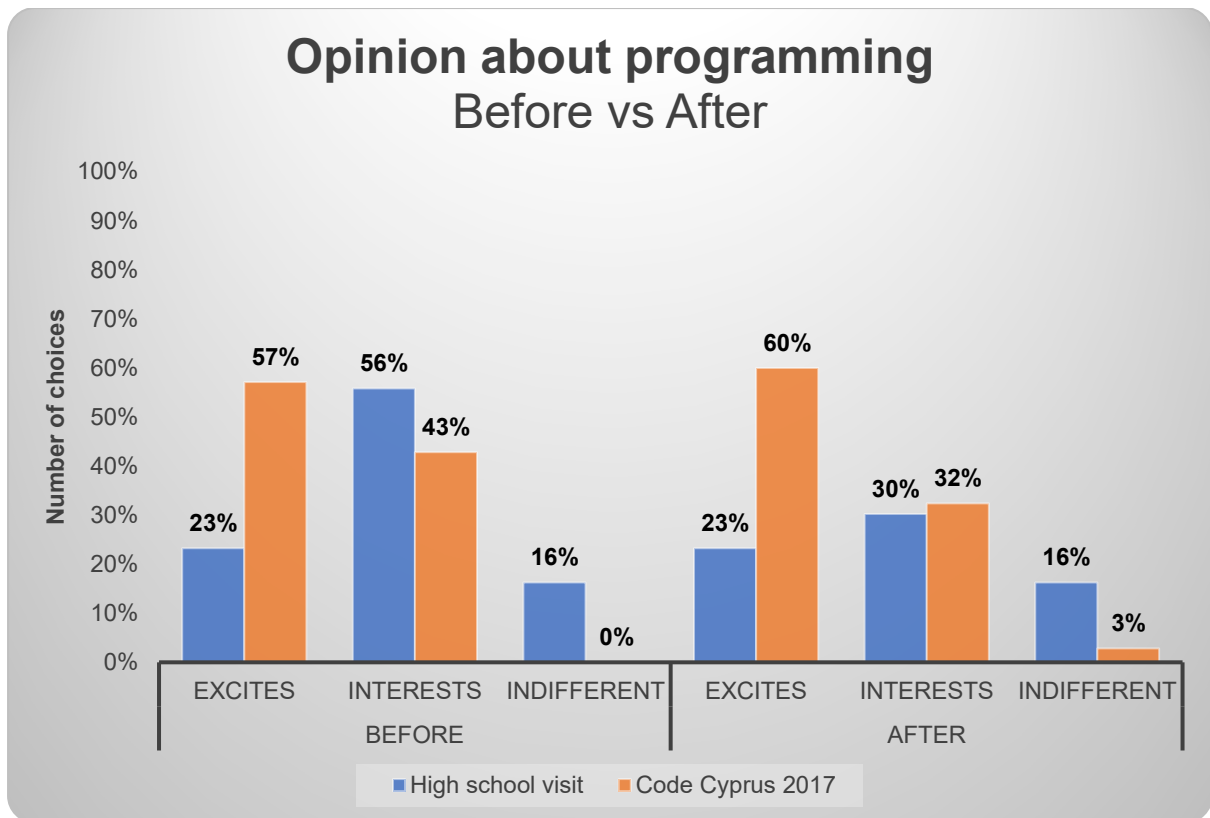


Figure 34 - A bar chart showing the differences between opinions on programming before and after the two events

9.22 Appendix 22

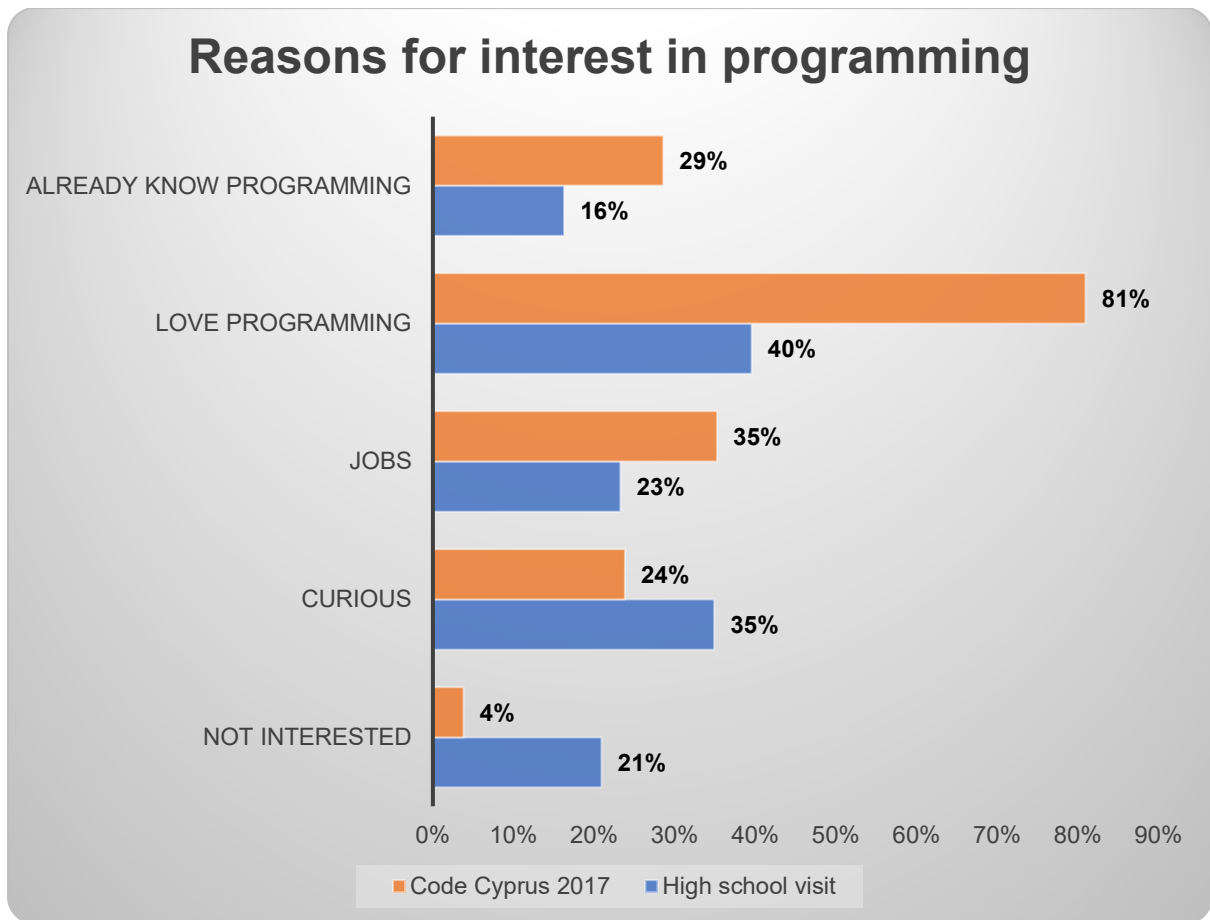


Figure 35 - A bar chart showing reasons for interest in programming by students.

9.23 Appendix 23

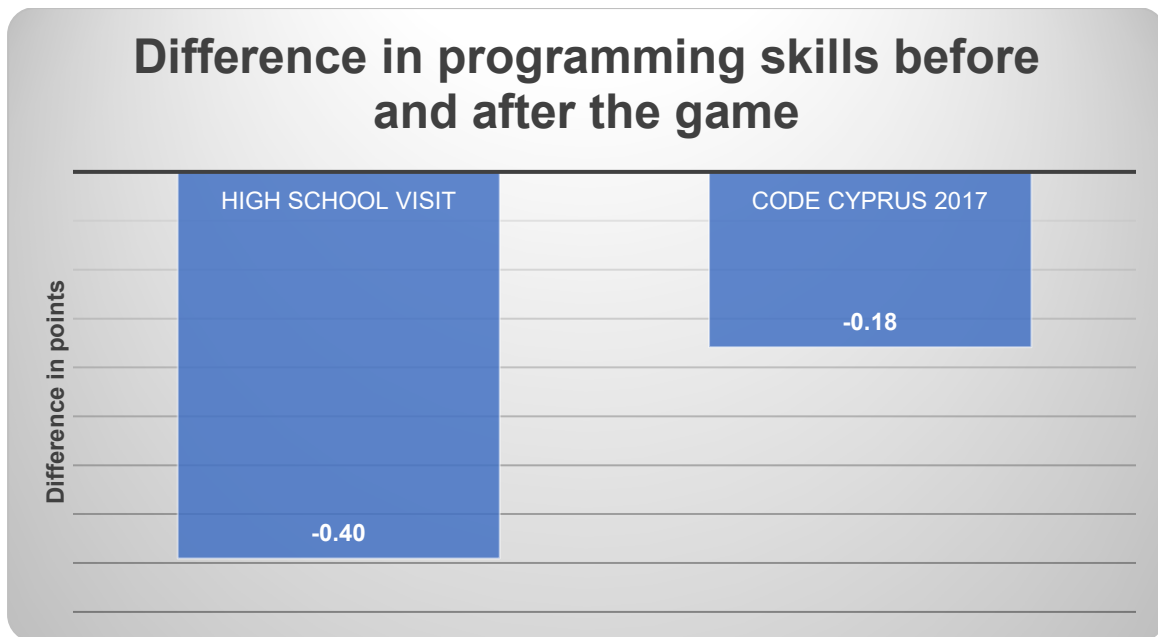


Figure 36 - A bar chart showing the difference in programming skills before and after the game.

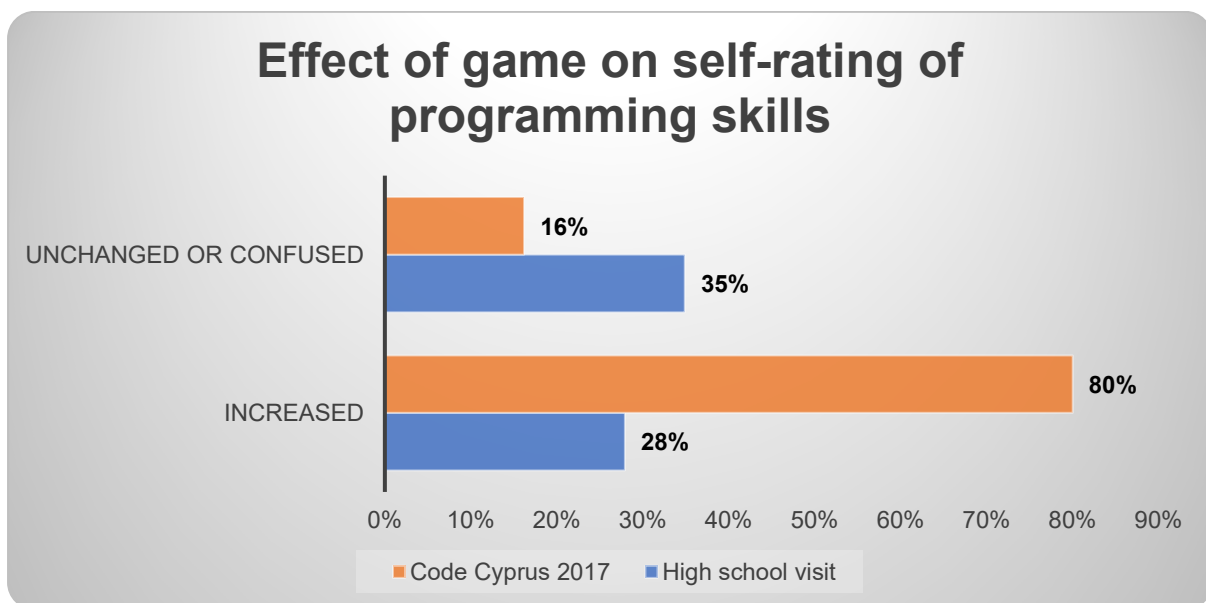


Figure 37 - A bar chart showing the student's perceived rating on their own programming skills after the game.

9.24 Appendix 24

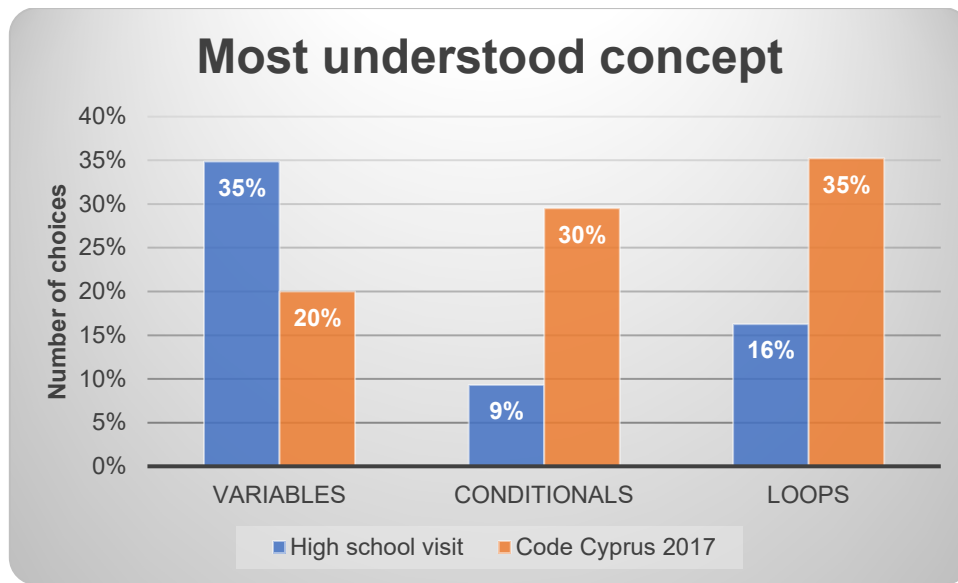


Figure 38 - A bar chart showing the most understood concepts by students.

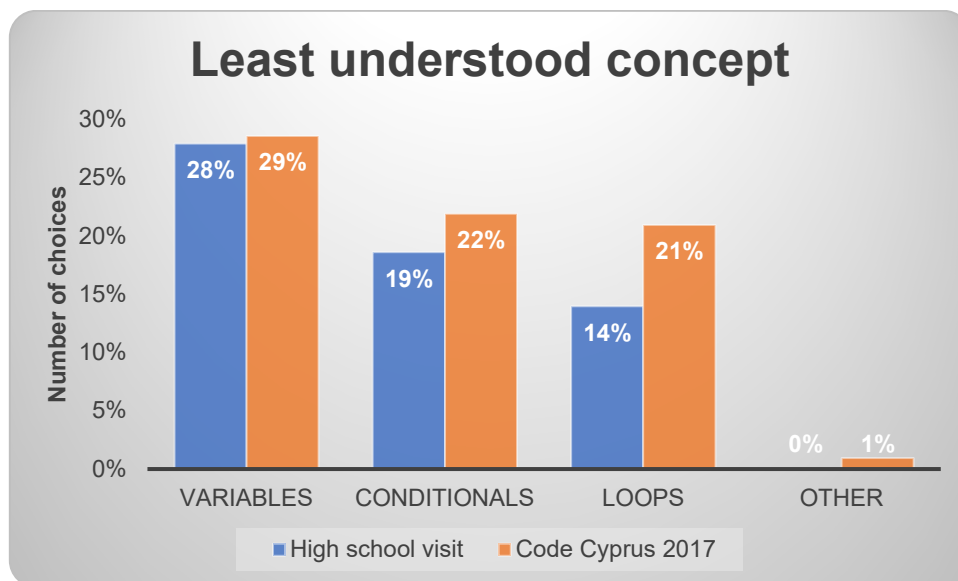


Figure 39 - A bar chart showing the least understood concepts by students.

9.25 Appendix 25

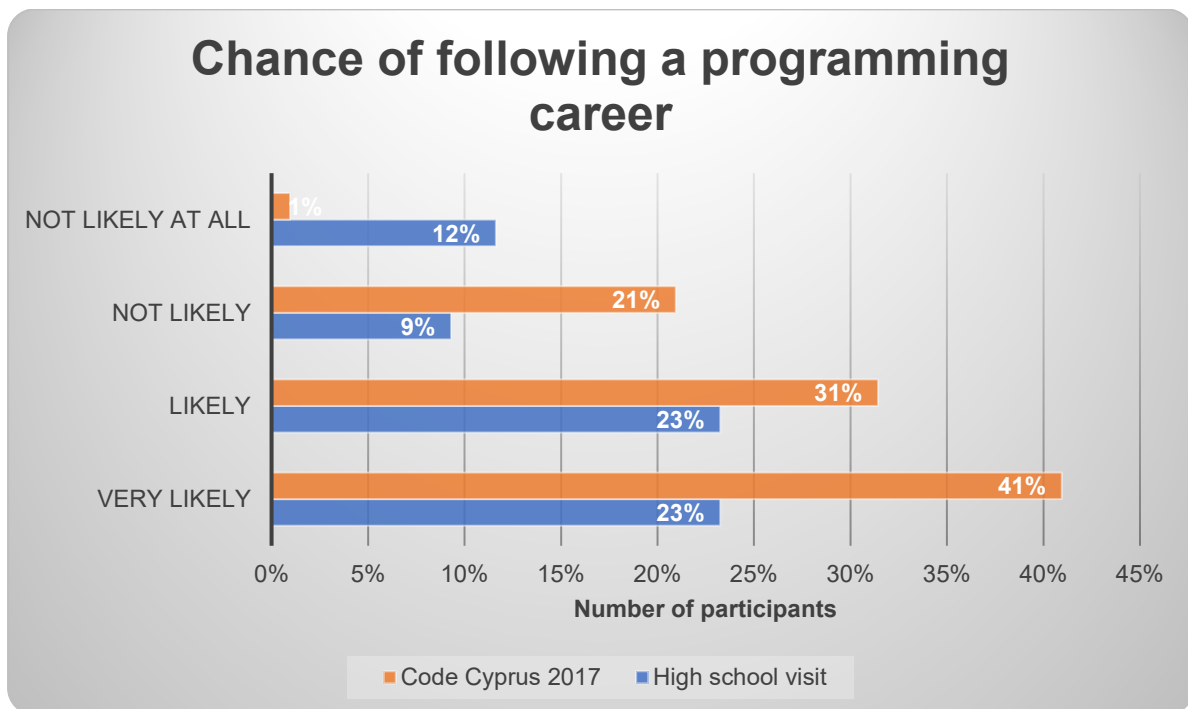


Figure 40 - Chances of following a programming career.

9.26 Appendix 26

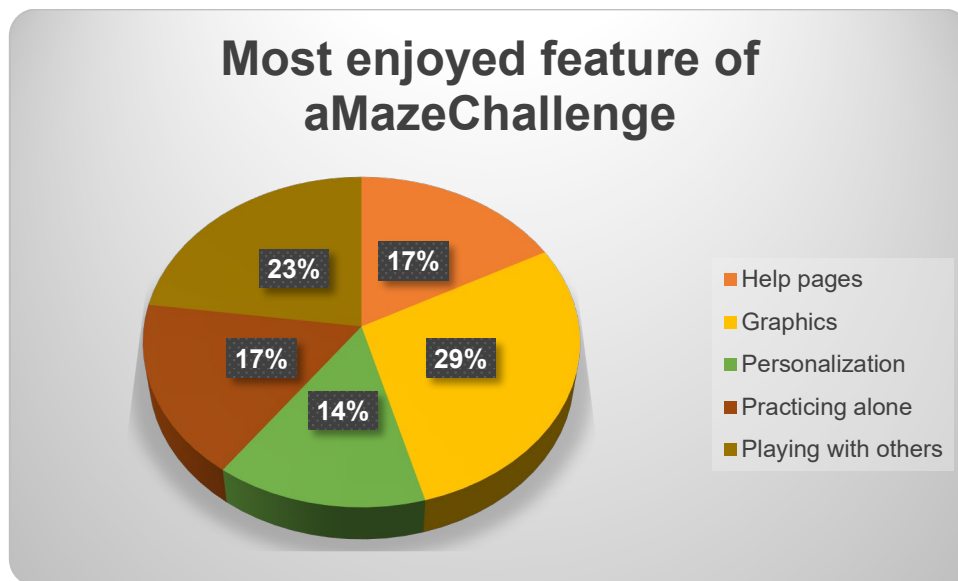


Figure 41 - A pie chart showing the most enjoyed features of aMazeChallenge

9.27 Appendix 27 – Project Proposal

BSc (Hons) Computing – Double Project Proposal

Name: Nicos Kasenides

Specialization: Software Engineering

Size: Double

Discussed with (lecturer): Dr Nearchos Paspallis

Current Modules (and previous modules if computing or direct entrant)

Introduction to Programming, Algorithms and Data Structures, Discrete Mathematics, Academic Writing. Computing Skills, Practitioner Skills, Introduction to Networking, Games Concepts, Interactive Applications, Systems Analysis and Database Design. Advanced Programming, Software Engineering Practices, Computer Security, Professional Skills, Software Development, Mobile Computing. Advanced Software Engineering Techniques, Object Oriented Methods in Computing, Enterprise Application Development, Computer Graphics, Double Project.

The Project Title: aMazeChallenge

Project Context

The aMaze Challenge is an educational Android-based game which aims to teach programming to young players, by allowing them to create their own code to escape a virtual maze arena. The project aims to teach the fundamentals of programming to a beginners audience through an immersive and fun environment. According to a study conducted by MIT, *“games can engage players in learning that is specifically applicable to schooling”*. An interactive game as such may excite a younger audience’s interest in learning not just programming but also science and technology, thus attracting more people to these fields which will be illustrated as fun and enjoyable. Lastly, the competitive nature of the game aims to engage the players in a skirmish from which they will hopefully grow increasingly interested to improve their programming skills and appreciate the benefits of computing. The project will be evaluated in the context of Code Cyprus 2018.

Specific Objectives

Must Have:

The game should be implemented in a cloud-based system (such as AppEngine).

The game should provide a view which visualizes the game progress and the player's standing in real time (a view of all the players' location and a live scoreboard) on an appropriate platform such as a web application.

The game should provide a mobile view which allows players to register and take part in the game by defining their code and viewing their progress. Implemented on an appropriate platform such as a Web or Android application.

An administration page with which the managers of the game will be able to monitor progress, view all data related to the game and generate new mazes for new challenges.

Should Have:

Realize engaging game elements such as graphics and sounds to enhance the experience of the players.

Include several other features such as dropping timed bombs, traps, health or time pickups etc.

Could Have:

An offline tutorial mode in which a player may practice their coding and test their algorithms against an AI opponent.

Won't / Would Have:

Video tutorials on how to play etc.?

Potential Ethical or Legal Issues

Since the game involves an audience which is predominantly under 18, special care should be taken so that none of the players are exposed to harmful situations or unpleasant material. In addition, the challenge serves as a means of gathering useful data about the learning capacity of the players in terms of programming. In case of any shadowing the players should be aware that their actions are being monitored and if they have to complete any forms, this should be done in a discrete manner to avoid any unnecessary personal data misuse.

Resources

No specialized equipment is needed for the development phase. The necessary tools (such as Android Studio and IntelliJ IDEA) are freely available, and the targeted platform (i.e. Google Play Store and Google AppEngine) provide free deployment (at least for low traffic apps). Third party libraries, such as Google Blockly and Mozilla Rhino are freely available for reuse via an open-source license.

The project will be tested as part of the next Code Cyprus event which is planned to take place at UCLan Cyprus in March 2018. We will need a projector to display the live scoreboard on a bigger screen, as well as a room/theatre in which the players will be gathered to register and take part in the event.

Several mobile devices will be needed for testing in addition to an Android emulator to make sure that the application runs correctly on several devices.

Potential Commercial Considerations - Estimated costs and benefits

The project has no tangible benefits. However, a large gap currently exists between the demand and supply of programmers and software developers/engineers in the job market. This is perhaps because a lot of people, especially children feel sceptic about programming and computing mostly because of its seemingly cryptic nature which makes it hard to understand at a young age. This game will demonstrate that programming is not just the writing of incomprehensible text but also a very powerful tool with which someone can explore their creativity, produce useful things and have fun. Furthermore, it will serve as a reminder of some fundamental principles related to programming for any players who have prior programming experience and hopefully engage them even more so that they improve their coding skills.

Proposed Approach

Create a basic Android application – using Android Studio, Java Graphics and the Blockly framework. Approximately 3-4 months.

Use existing or create a new code interpreter - used for interpreting the code written by the players as actions in the game. Will take approximately 2-3 weeks for existing interpreter and 2-3 months for a new code interpreter.

Finalize a more advanced Android application – which will utilize more features such as bombs, traps etc, an improved user interface and allow users to register using their e-mail accounts.

Create the server API and database – used to store data related to the game and to allow the Android application to interact with the server. This will be implemented with and deployed on Google AppEngine and Java Servlets and will take approximately 1-2 months.

Create the administration / management page – Will be implemented using Google AppEngine and the MaterializeCSS framework. Needs to be robust and reliable. Approximately 2-3 weeks.

Testing Android Application – This should be done in offline mode and on various devices of variable screen size and other properties. Should take approximately 1-2 weeks.

Integrated game and platform testing – Test the entire, integrated game (Android app, Server API, Administration page) for any problems and bugs and suggest improvements to usability and user experience. Should be rigorous and utilize a higher number of people and a wider range of devices, browsers and platforms for compatibility testing. Should take approximately 2-3 weeks up to the Code Cyprus 2018 event.