## Analysis

### Analysis - Problem Definition

There is a severe lack of quality arcade games in the current market. Arcade games such as Pac-man, Space Invaders, and Street Fighter became cult-classics not only for those who play games, but for the public due to their ingenuity, competitiveness and their simple premise during the arcade era. Arcade games are, however, largely forgotten now in exchange for Triple-A console games supported by large companies. The revenue of the arcade game industry dropped to $1.33 billion in 1999, and then subsequently reached a low of $866 million in 2004 due to the more prominent home consoles at the time [1]. By the early 2000s, the prominence of these home consoles and networked games replaced the once vibrant atmosphere exclusive to the arcade market in the west [2]. The only remnant of these western classic arcade games exists in the mobile market, where cheap, simple games are made with no real substance or aesthetic to them. Due to this decline in arcade games, I want to create a product that pays homage to these acclaimed arcade games of the 1970s; a game that targets those that love the style of these cult-classics.   
  
The end user for my product must share similar preferences to those who loved arcade games before their decline (as my product will ultimately be inspired by these arcade games). In order to investigate the preferences of my end-users, however, I must first identify the characteristics of my end-users. This can be done by researching the demographics of existing arcades in Japan (a country where the arcade industry has remained popular into modern times) and by looking at past demographical data based on the golden age of arcades.

In the 1990s a survey was conducted by […] showing that 70 percent of all arcade players in the US were teenage males who were dependent on a guardian or parent. This demographic was said to have had a high skill set in playing games, preferring fast-paced, competitive games more so than slow ones. This demographic also had a preference for local multiplayer games, proven by the popularity of fighting games such as Street Fighter and Mortal Kombat. These end-users were ultimately reliant on the social interaction that comes from arcade games [3]. Based on a study by […], the average US teenager was likely to be in high school education.

Existing Japanese arcades share similar findings. Taito Hey, one of the most popular arcades in Japan, is dominated with shoot ‘em up, rhythm and fighting games such as Street Fighter IV, Dance Dance Revolution X2, Mobile Gundam vs. Gundam Next, and more. These games require a quick reaction time and have a competitive component to them, similarly to the western arcades during the 1990s. On the other hand, the Japanese market differs from the US in terms of demographic, with [...]

The idea of making an arcade game is not enough to satisfy the need for my end user however. To create an arcade game, I need to have a premise that’ll act as the foundation for my product. One idea I had was to create a maze game in the vein of Pac-Man that incorporates a competitive aspect to satisfy my end user.

Mazes can help develop spatial navigation skills, memorization, and […], according to a study done by [...]. In the 1970s, there was a maze craze in which many magazine publications and books devoted strictly for mazes were commercially made available. Well known authors include Vladimir Koziakin, Rick and Glory Brightfield, Dave Phillips and more. The wide number of publishers will allow me to siphon through to get the best ideas. The video game industry has had multiple well-known games that involve some form of pathway or labyrinth, such as Spectre, MIDI Maze, and most notably Pac-Man. A labyrinth/maze game with an arcade aesthetic, competitive multiplayer and a fast-pace is likely to meet all of the credentials for my end-user, particularly those who are in their teens.

Ultimately, the issue I want to solve is the lack of quality arcade games in the current market and provide end-users who missed the golden age of arcade games an opportunity to play something similar on a modern device. My proposed solution to the issue would be an arcade maze game where each player goes against one another (or the AI) in an attempt to beat their opponent in a race to their own respective exits. Like arcade games, the game will feature simple and intuitive controls and short stages against the AI which increase in difficulty. These features will satisfy the need for my end user as they’ll successfully help replicate the style of an old arcade game. Questionnaires and interviews will be done to further investigate the problem.

The main issues that I want to address include:

* Fixing the lack of arcade games in the current western market.
* Creating a fast-paced game that targets the current difficulty and complexity of current games.
* Creating a competitive game.

Potential preferences of my end user include:

* A product that is fast paced and requires quick reaction time
* A product that includes some form of competitive multiplayer.
* A product that includes AI interaction.

# Analysis – Justification of Computational Methods

To create a fast-paced, competitive maze game with an arcade aesthetic, a computational method is preferred.

A non-computational solution to solve the lack of maze games may include a bespoke real-life maze where the maze walls/junctions could be swapped in order to create a new maze each time. An ordinary maze would also suffice as a non-computational method. The issue that lies with these non-computational solutions is that whilst these non-computational methods would help satisfy the need for maze fanatics, these methods do not satisfy the need for arcade enthusiasts, nor do they provide a fast-paced, competitive environment easily. To provide the arcade aesthetic that my end-users want, a computational approach is necessary as the arcade aesthetic is intimately linked with computers (e.g. a CPU cannot be created non-computationally, fast procedural generation cannot be done non-computationally, scoreboards cannot be created non-computationally, etc.)

The benefits of a computational method (and the issues with a non-computational method) are as follows:

**Physical Involvement:**

Physical involvement is a necessity when participating in either a non-computational maze or an arcade. This necessity makes participation difficult for users with physical disabilities to play an arcade game or explore a maze as they wouldn’t be able to play or explore them out of their own volition. This necessity of physical involvement also makes it difficult for parents with children as the parents would have to keep eye on their children in an arcade and especially in a maze, where they could get lost. Travelling to these non-computational venues would also be an issue due to the travel cost. A non-computational arcade/maze would have to act as a local business instead of having the global support of an app store. A labyrinth/maze arcade game made computationally and made available in the app store would help alleviate the issues stated above: those with physical problems or children would be able to participate in the game in the comfort of their own home and a computational mobile approach would have the support of a global app store, reducing travel.

People may argue that, with my target demographic being between the ages of 12 to 18 years old, the probability that my user will be physically disabled is less likely. Only 7% of children in the UK are disabled, compared to 18% of working age adults and 44% of adults over the state pension age [1]. This brings the question of whether a computational maze is truly necessary to solve the issue of physical involvement, particularly a game that is primarily targeted at the adolescent demographic. I would argue, however, that the physical involvement necessary to traverse a non-computational maze or travel to an arcade cabinet is still a major issue considering that 7% of children equates to a population of roughly 800,000 within the UK. I would also argue that children are less likely to travel by themselves unless accompanied by an adult, bringing up the issue of taking care of children in a maze once more. The issue of travel cost is also still unsolved.   
  
Ultimately, the ease and convenience of being able to participate in a maze at home outweighs the potential time and cost spent making a computational method.

**Costs and Safety:**

In a non-computational approach, arcades and mazes must incur maintenance costs to ensure that they are kept in a good condition for the public. This leads to an economic loss for the majority of arcades/mazes as they typically lose more money than they bring in, according to a study run by […]. The decline in the arcade industry demonstrates a noticeable risk in starting an arcade business. In general, there’s a higher risk involved when creating a real-life arcade/maze as that approach as a higher initial investment cost when compared to a computational approach. A computational approach can have little to no maintenance cost or initial investments costs than a real physical business, making it a safer investment overall.

In addition to cost, a non-computational arcade/maze is more dangerous than a computational one. People can get lost in a non-computational, real-life maze, leading to general endangerment – particularly for 10 to 16 year olds. Because of this danger, customer service, admissions, and security will have to be available at all times to ensure that customers are attended to and criminals aren’t allowed in, making that initial investment cost even higher than before.

Maintenance cost, customer service cost, safety, etc. isn’t an issue with a computational approach due it being digital program that the user can download, thus making a computational game a better alternative in terms of maintenance costs and safety.

**Competition + Perspective + AI:**

A non-computational maze is unlikely to solve the need for competition for my end-users due to the little involvement a real-life maze requires. At a first person perspective, a person cannot avoid dead ends or estimate the location of an end point in a maze. The user must randomly go through junctions to complete the maze. This random aspect makes it difficult to improve in the game and thus difficult to compete in, making that solution inappropriate for my end-users. To compete against others in a non-computational maze, users are also required to have other people physically present in a non-computational maze to satisfy their need for competition, making it more difficult to set up than otherwise.

A computational maze can successfully apply elements to raise the skill level of the game, satisfying the end user. Scoreboards, player vs. player interaction, modifiable difficulty levels, and – most importantly – a bird eye’s perspective – can help increase the skill level of the game, leaving the end user more satisfied than they would be for a non-computational maze. A computational maze can create computer-controlled opponents through the use of maze solving algorithms, ensuring that the competitive element of the game is always there. This application of game-design elements and principles would help satisfy the need for competition in my end-users and improve user engagement with the product.

**Algorithms + Random Generation:**In a non-computational maze the layout cannot be reconstructed or changed easily. The layout would have to be changed manually by either robots or actual people, leading to generally higher maintenance and working costs. This limitation prevents the variation and excitement that would come with exploring a maze with new dead ends and new pathways. This approach also limits the competitive element of the maze.

With the use of computation, however, a maze can be randomly generated. This can be done by using the recursive backtracking algorithm to figure out the desired path (using each junction as a node), and then adding new pathways to simulate dead ends. The recursive backtracking algorithm (and other maze generating algorithms) enables the fast production of new maze layouts, ensuring that the user is always invested and interested in the game. This could not be done in a non-computational approach as a non-computational approach cannot use computational algorithms successfully.

**Abstraction:**

Abstraction can be used to make a computational solution more efficient. By simplifying a problem to its simplest components, we’re able to reuse models that are common in the program (e.g. assets, menus and images could be reused). This could not be done in a non-computational approach as non-computational elements are not reusable; they have to each be physical and tangible on their own.

**Analysis – Suitable Stakeholders**

Who are my current stakeholders?

My current stakeholders are people who like to engage in playing competitive, arcade games. My stakeholders are likely to be between the ages of 12-18, and are likely to be proficient in other competitive games. The stakeholder is also likely to be in education given that the majority of people between the ages of 12-18 in the UK are in education. The stakeholder is also likely to be proficient in the use of mobile devices and social media. As of 2012 at least 70% of 12 – 17 year olds reported having at least one social media profile.

One of the main needs of my stakeholder is social interactivity. Communication with peers and family members increases drastically between the ages of 12-18, and peer relationships tend to become deeper than otherwise during this period of time. This social interactivity would be provided through my product by implementing local multiplayer features, allowing my stakeholders to play against one another and socially interact with their peers through a game. Online games, particularly competitive ones, enable the formation of lasting relationships by encouraging further positive interaction between players after the game is complete.

Another characteristic my stakeholders share is an interest in interactive games. Interactive games are stereotypically associated with males due to how prevalent and receptive the male demographic were to these forms of entertainment – particularly arcade games. However, recently the diversity of the audience has been increasing for those who play video games. The competitive fighting game scene in particular is noted to be racially diverse and more tolerant than other communities. The source of this diversity is commonly attributed to its origin in arcades, where audiences where able to compete against people of different backgrounds in a positive social environment, and where the barrier of entry was low. This diversity in games, particularly competitive games, indicates that my product does not have to target any particular male demographic, and that producing a fair competitive game is likely to satisfy the need for a lot of my stakeholders.

How will my current stakeholders make use of the current system?

My stakeholder would make use of the current system through the use of PC or a mobile phone. They will be two versions depending on where the stakeholder would prefer to play the game. The stakeholder would make use of the product preferably alongside another one of their peers, where they would be able to competitively play against one another in a series of procedural generated mazes. The user could, however, play the game on their own against a CPU opponent if they wished. On a PC or laptop, the stakeholder would be able to play as Player 1 using the WASD keys, whilst their peer would play as Player 2 using the arrow keys on the same keyboard. On a mobile phone, the multiplayer feature would work using a P2P system where the stakeholders connect their phones to each through the app. On the mobile phone version, the user would be able to use touch screen controls that sit on the corner of their screen. The current stakeholders would preferably make use of the current system at their own home.

**Analysis – Current System**

In order to further understand the needs of my end-users, I have decided to investigate the games that have similar characteristics to my proposed end product. In the investigation, I aim to identify as many problems with the current system, and evaluate what characteristics I must avoid in my own product. I also aim to identify the positive traits with each current system, to see what features I could add to my own product.

The Amazing Maze Game

One of the existing systems I decided to look at was the original *The Amazing Maze Game*. Released in October 1976, this maze game was received with a lukewarm reception in spite of its maze complexity and coding. It was released in an upright dedicated cabinet and it was placed in classic arcade stores.

The Amazing Maze game had numerous amounts of features for what it was at the time. It had mazes that increased in complexity, AI vs. Single Player interaction, local multiplayer and randomly-generated pathways. This makes it one of the best examples to compare with as it has many of the necessary components I want to implement into my product. In addition, it’s a good comparison to see what issues I must avoid in my own product.

One of the main issues with The Amazing Maze Game was how boring each match was. I found that by playing the game myself I got tired of each match quickly, due to the lack of speed and precision provided by the game. The maze was composed of long pathways, making each match a tedious walk to the end-point, as shown by Figure 1. A way to overcome this issue would be to increase the walking speed and reduce the sizes of the maze so that the player can successfully win in under a minute. This adaption will make the game more fast-paced, increasing user involvement and making the game less boring overall.

Another issue with the game was the imprecise controls. One review of the game criticised the controls, stating that “you’ll have to struggle a bit for some turns”. The Amazing Maze Game’s maze layout and character controls seemed to prevent the user from developing the necessary speed to beat the computer. Watching a play through on YouTube, I noticed that players were unable to get through certain junctions, as they would have to thread their character/block carefully through each pathway. This threading limited speed and made it difficult for the user to traverse the maze. An intuitive control scheme is vital for competitive games, and it’s an aspect that I must review regularly beforehand to prepare for development. A method to solve this issue would be to implement a script that prevents this threading from happening.

Tomb of the Mask

The next existing system I wanted to look into was Tomb of the Mask. Released in February 2016, this modern arcade-like mobile game was well-received by the public for its “retro FX” and “fast-paced gameplay”. Tomb of the Mask has a few similar aspects to my proposed solution (e.g. multiple pathways, arcade inspiration). Due to this similarity, I decided to research Tomb of the Mask. Tomb of the Mask has a clear appreciation for arcade games whilst providing the user with a tight control scheme and a fast pace. Researching this game is likely to give me ideas on how to modernise my arcade game for modern audiences whilst providing that fast competitive element.

One of the main compliments of the game I have seen for Tomb of the Mask was its controls. Despite the fast-pace, the game was responsive and allowed others to gain momentum through each change in direction. This fast pace was contrary to The Amazing Maze Game, where the users was unable to gain momentum successfully. Whilst The Amazing Maze Game’s controls were dependent on holding a button, Tomb of the Mask’s controls was dependent on swiping the character in that general direction. The character in Tomb of the Mask only stopped when the object had a collision against the wall. This made movement easier in Tomb of the Mask as the user didn’t have to slow the character object themselves down; all they had to do was move.

The main issue I found with Tomb of the Mask was in its menu design. Random ads popped up from time to time, disrupting its potentially fast re-playability. The menus go against the philosophy of many arcade games which have simple menus to ensure the user keeps playing. This issue of menu design was unavoidable for this game, however, as the game is free in the App store and requires some form of income due to its development.

Another problem with Tomb of the Mask is the lack of competition. The need for a competitive environment was an essential characteristic of my end user. The lack of competition in Tomb of the Mask makes it ill-suited to solve the issues of my end-users as there weren’t any multiplayer or CPU modes. One of the methods Tomb of the Mask uses to create a competitive environment is having an endless mode, where the player has to navigate around obstacles in a vertical labyrinth to avoid death. This successfully produces a competitive environment for the player against the labyrinth itself, but it does not provide that competitive element against other people. Whilst Tomb of the Mask replicates arcade games in terms of aesthetic, it does not replicate arcade games in terms of multiplayer gameplay nor menu design.

Current System Flowchart

Control Scheme + Speed + Maze Design

By researching Tomb of the Mask and The Amazing Maze Game, I have realised that the control scheme of a product is essential in ensuring the user is satisfied. I found that by playing The Amazing Maze Game I was left feeling dissatisfied due to the inability to build up speed. In The Amazing Maze Game, I would have to manually stop at certain junctions and thread my character through junction gaps, unlike Tomb of the Mask where I could successfully build up this speed by swiping the screen. This issue was mostly a result of cornering mechanics present in The Amazing Maze Game

By doing further research online and reading forums I have noticed a number of instances where others have had issues in solving the issue of cornering within 2D maze games. This mostly comes as a result of the colliders (the character and the wall) interacting with one another either too early or too late. This prevents the user from moving successfully and quickly in the maze. Leaving an issue like this in my product would mean that the user would not be able to build up speed in the game successfully. This would, in turn, fail one of the requirements in making a fast-paced, competitive game.   
  
A solution to this problem would be to implement a unicursal maze design (a labyrinth) instead of a multicursal maze design. This would allow the user to move their character quickly throughout the maze without having to thread the character through junction gaps (this maze design removes all T-junctions). This design would ensure the game remains fast-paced for all users. Tomb of the Mask’s control scheme (where the character stops moving after they collide into a wall) would work well with a labyrinth design, as they both complement fast gameplay with no threading. This design, however, would heavily compromise the variation in maze design due to the lack of dead ends. Due to this compromise, I am wary on using this solution. In testing and interviews, I should evaluate whether this sort of maze design would be suitable for the end-users.

Another solution to the problem would be to implement a script where the character object has to stop at every T-junction in a maze. This differs from a unicursal maze as the maze design isn’t compromised for the sake of control and speed. Instead, the maze design is able to maintain its complexity whilst allowing the character to move fast. This would be done by taking in a list of nodes that have neighbours equal to the number three and using that list to implement pass-through colliders. This solution is personally my preferred solution but I’ll need to evaluate what my end-users think of beforehand.

Complexity of Menu Design

In addition to the control scheme, I have also realised that a simple menu design of a game is essential in ensuring that the end-user is satisfied with the results. Tomb of the Masks’ menu design was disruptive due to their insistence in placing ads, and the transitions between each menu option were annoying – making the overall experience when playing the game worse. A complicated menu design would not be appropriate for my solution as a complicated menu design would make it difficult for users to immediately play the game. Arcade games typically have simple menu designs so that the user can quickly play the game again if they wish. Due to this, I’ve decided to implement simple menu designs more so than complicated ones.

Conclusive Analysis of Current Systems

By researching each of the respective games more I have realized that the proposed solution I have come up with has a couple of obstacles ahead of it, with the biggest one being the controls. I decided that my solution should be fast-paced, yet I also decided that it should be a maze — with varying pathways. These two aspects juxtapose one another as a high speed makes it more difficult for the user to go through multiple pathways in the game, and the multiple pathways makes it difficult for the user to build up speed. This can be solved however by creating a control scheme and a game design that complements fast pace.

### Analysis – Investigation

Preparation for Investigation

To begin the production of my product, I must first clarify what my end users’ want out of their current system in detail. This can be done by producing an investigative report that explores the reasons why the end-users dislike/like the current system. This investigative report will help me modify my proposed solution accordingly and it’ll ensure that my proposed solution targets all the necessary problems of my client. It’ll also help me discern the limitations of my product to see what pitfalls I must avoid.

To complete the investigative report, I will create several questionnaires that target the necessary pieces of information I am trying to find. These questionnaires will differ depending on whether the person has played Tomb of the Mask, The Amazing Maze Game or neither. After the questionnaires have been complete, I’ll collate the results from the questionnaires and investigate them. I’ll also ensure that every participant in the questionnaire signs a form to act as further proof.

The questionnaire will be given to pupils from The Langley Academy, ranging from the ages of 12 to 17. These pupils are a suitable demographic to ask opinions as they share the general age range of my end-users. In addition, their close proximity to my location will allow me to get more information/results in less time due to lower travel costs, improving the general reliability of the investigation.

Key Pieces of Information

The key pieces of information I need are as follows:

* Menu Design
* Maze Design
* Control Scheme
* Pace

I decided on these elements as they are essential in providing the product that I want to give. The menu design and control scheme are essential in providing the client with the good UI. The maze design and pace are essential in ensuring that the feel and the aesthetic of the product are good.

The questions below will help me get the key pieces of information that I stated above:

* Did they like/dislike the controls of Tomb of the Mask?
* Did they like/dislike the controls of The Amazing Maze Game?
* How did they find the pace of Tomb of the Mask?
* How did they find the pace of The Amazing Game?

These questions can be answered by having a sample of the end users play either The Amazing Maze Game or Tomb of the Mask for five minutes, after which they’ll complete a questionnaire describing what they like and dislike about the game. The results of the questionnaire will help finalise my decisions on the maze layout, control scheme, multiplayer, and more.

**Questionnaire – General**

The questionnaire below will act as a general quiz for those who have not played Tomb of the Mask or The Amazing Maze Game. This questionnaire will be given around to 20 people.

The objective of this preliminary questionnaire is to collect background data to determine the direction of our arcade game. Your opinions and choices on the following questions will help determine the shape of the game.

Please answer the following questions by circling the appropriate option.

Would you prefer a fast-paced, medium-paced, or slow-paced game?

Fast-paced Medium-paced Slow-paced

Which maze design do you prefer?

**Results of “Questionnaire – General”**

Would you prefer a fast-paced, medium-paced, or slow-paced game?

70.8% of people decided that they would prefer a fast-paced game. This result coincides with my original proposal in creating a fast-paced game and it aligns with the wants of arcade fanatics. This question demonstrates the appeal of fast-paced game in adolescents, and it provides evidence to the idea that my proposed solution will have a wide appeal.

Which maze design do you prefer?

45.8% of people decided the Maze Design B was the better one. There were, however, a number of comments that stated there were not enough options to pick the best maze. One person, after finishing the questionnaire, told me that they “liked things about A and things about B” but they had to decide between them. Due to these comments, I have decided to create a revised questionnaire further down based solely on this question. The revised questionnaire will attempt to target some of the initial concerns from this question and get a more conclusive result.

Would the inclusion of a scoreboard be suitable?

79.2% of people decided that the inclusion of a scoreboard would not be suitable for the game. When asked to elaborate, they said that the inclusion of the scoreboard wouldn’t actually incentivise them to play more – as the game idea itself is already sufficient. Others went on to say that the inclusion of a scoreboard would clutter the game and that they’ve never really found a use for scoreboards in modern games. The inclusion of a scoreboard is a staple of the arcade genre, thus I've decided to implement it in spite of this result to better replicate the arcade aesthetic. I will, however, try to limit the functionality of the scoreboard to make it as minimalistic as possible during development.

Which menu system do you prefer?

(Just to elaborate, I decided to remove Tomb of the Mask and The Amazing Maze Game from the potential menu options as they both had poor menu layouts. For this question I instead opted to include menu designs that I came across during my research. These menu designs generally go from simple to more complex.)  
  
66.7% of people decided that they preferred Menu Design C. Comments stated that the “simple design” of Menu Design C was their reason for choosing that over the others. Other comments stated that it was easy to understand Menu Design C and it had colours that made it “pop”. The simple minimalistic aesthetic of Menu Design C will aid in creating the arcade aesthetic that my end-users prefer and it help in creating the overall design of my maze.

**Analysis/Evaluation of “Questionnaire – General” Results**

The results of the general questionnaire have helped me decide on which menu design I should implement and whether the inclusion of a scoreboard in the game would be suitable or not. The general questionnaire has also helped reassure me about the potential of a fast-paced game in the market and shown that samples of 13 to 17 year olds in The Langley Academy are positive about the concept of my product.   
  
**List of Key Points from “Questionnaire – General”**

* Participants prefer simple menu designs more so than complex ones.
* Participants stated that the inclusion of a scoreboard would not be necessary for the game.

**Revised Questionnaire – General**

The questionnaire below is a longer modified version of “*Which maze design do you prefer?”* The revised questionnaire has more maze design options as well as a question asking why the user prefers that maze design in comparison the rest.

The objective of this preliminary questionnaire is to collect background data to determine the direction of our arcade game. Your opinions and choices on the following questions will help determine the shape of the game.

Please answer the following questions by circling the appropriate option.

Which maze design do you prefer?

Would you prefer a long maze or a short maze?

Long Maze Short Maze

Please write any additional comments below. Thank you.

**Results of “Revised Questionnaire – General Results”**

Which maze design do you prefer? + Why do you prefer that maze design?

[...]% of people decided that Maze Design E was the best one. They preferred Maze Design E due to bold outline on the walls and its simplicity in comparison to the other mazes. The unicursal element of Maze Design E, in addition to the bold outline, made it a clear winner in comparison to the other mazes.

On the other hand, a substantial percentage of people decided on Maze Design B ([…] %). This group stated that they preferred this maze design due to its complexity in layout, preferring the multi pathways more than the simplicity of Maze Design E.

Would you prefer a long maze or a short maze?

[…]% of people decided that they’d prefer a short maze more than a long maze. A short maze design would complement and promote a fast-paced game, thus I’ll be sure to make a short maze. This can be easily done by removing a couple of columns and rows from the grid algorithm I’ll have to make in the future. One of the issues with having a purely short maze is that match times tend to be a bit shorter naturally. In addition, procedurally generated mazes may result in incredibly simple mazes being created every one in ten matches, more so with short mazes than long mazes.

**Evaluation of “Revised Questionnaire – General” Results**

The results of the revised questionnaire seem to align with my previous assumptions, though they have given further clarity and emphasis on certain aesthetic features. I hadn’t realised that the boldness of the exterior walls could influence a user’s preference. This aesthetic feature will require manually checking from time to time to ensure that it matches my end-users’ preference.  
  
Overall this revised questionnaire was better as it demonstrated that my clients have a preference for multicursal mazes. Because of this preference, I leaning towards removing unicursal mazes from my potential maze design as both the aesthetic (according to my clients) and complexity of the maze would decrease when using a unicursal design.   
  
**List of Key Points from “Revised Questionnaire – General”**

* Participants generally prefer multicursal maze designs more so than unicursal designs.
* Participants generally prefer bold mazes more so than thin walls.
* Participants generally prefer shorter mazes more so than long mazes.

**Questionnaire – Tomb of the Mask**

The questionnaire below is one that was given to people after they had played Tomb of the Mask for 5 minutes. This questionnaire was given to 13 people.

The objective of this preliminary questionnaire is to collect background data to determine the direction of our arcade game. Your opinions and choices on the following questions will help determine the shape of the game.

Using a scale of 1 = Poor to 10 = Excellent, please rate the following features of The Amazing Maze Game. For written questions, please answer with black pen.

How enjoyable was Tomb of the Mask in general?

1 2 3 4 5 6 7 8 9 10

Why did you/did you not enjoy Tomb of the Mask?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

How good was Tomb of the Mask’s control scheme?

1 2 3 4 5 6 7 8 9 10

Why did you/did you not enjoy Tomb of the Mask’s control scheme?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

How was the pace of Tomb of the Mask?

1 2 3 4 5 6 7 8 9 10

Please write any additional comments below. Thank you.

**Results of “Questionnaire – Tomb of the Mask?”**

How enjoyable was Tomb of the Mask in general? + Why did you/did you not enjoy Tomb of the Mask?

All 13 people rated Tomb of the Mask favourably, giving Tomb of the Mask an average score of 7.5 for how much it enjoyed it. Many of the users commented on the tight control scheme and the fast gameplay as being the reason for their high score. Other people commented on how the sprite and 8-bit design made the game unique in comparison to other mobile games. One response commented on how “the limited colours of the game made it pop”. This comment gave me the idea to make the proposed solution more minimalistic. Three of the comments mention how the menu system of the Tomb of the Mask was “bad” and how it made the overall experience a bit worse. However, the general response to Tomb of the Mask was positive.

How good was Tomb of the Mask’s control scheme? + Why was Tomb of the Mask’s control scheme good?

Again all 13 people rated Tomb of the Mask favourably, giving its control scheme an average rating of 8.5. People commented on the swiping mechanic as being the reason why they liked the control scheme while a few commented on the short input time. The comments for this question seemed to overlap a lot, with a lot of the participants commenting on the sliding mechanic at least once.

How was the pace of Tomb of the Mask?

The overall pace of Tomb of the Mask achieved an average score of 8.8. This high score demonstrates the fast-pace of Tomb of the Mask. The fast-paced seemed to stem from the fact that the four-directional movement of the character was consistent and continued until the character object hit the wall. This made it easier for the participants to move their character as they didn’t have to thread their character through T-junctions.

**Evaluation of “Questionnaire – Tomb of the Mask?”**  
  
The results of the questionnaire seem to align with my typical assumptions when rating Tomb of the Mask. People rated the overall gameplay and feel of Tomb of the Mask highly due to how the character controls, while others were disappointed with the complex menu design Tomb of the Mask had. Overall, the reception to “Tomb of the Mask” was generally positive with all participants, making it a good system to compare my end-product with during the evaluation phase and development phase.

One of the common themes I noticed throughout the results of the questionnaire was the wide appeal of the movement in “Tomb of the Mask”.  
  
**List of Key Points from “Questionnaire – Tomb of the Mask?”**

* Participants generally prefer controls with a short-response time.
* Participants generally prefer menu systems that are simplistic in nature.
* Participants generally prefer control schemes that need as little input as possible for the greatest amount of movement.

Questionnaire – The Amazing Maze Game

The questionnaire below is one that was given to people after they had played The Amazing Maze Game for 5 minutes. This questionnaire was given to 16 people.

The objective of this preliminary questionnaire is to collect background data to determine the direction of our arcade game. Your opinions and choices on the following questions will help determine the shape of the game.

Using a scale of 1 = Poor to 10 = Excellent, please rate the following features of The Amazing Maze Game. For written questions, please answer with black pen.

How good was The Amazing Maze Game in general?

1 2 3 4 5 6 7 8 9 10

Why did you/did you not enjoy The Amazing Maze Game?  
  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

How good was The Amazing Maze Game’s control scheme?

1 2 3 4 5 6 7 8 9 10

Why did you/did you not enjoy The Amazing Maze Game’s control scheme?  
  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

How fast-paced was The Amazing Maze Game?  
  
  
1 2 3 4 5 6 7 8 9 10

Please write any additional comments below. Thank you.  
  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_: Signature

**Results of “Questionnaire – The Amazing Maze Game”**

How enjoyable was The Amazing Maze Game in general? + Why did you/did you not enjoy The Amazing Maze Game?

The Amazing Maze Game was rated an average score of […] by the participants. The scores and comments of the participants were generally mixed, with a few commenting on the poor controls and boring, long matches as their reason for a low score. Others could see potential in the game, stating that “playing against the CPU was long but still fun regardless”. The positive reviews of the game, for most participants, seemed to stem from the maze itself, with participants stating that the variability of the maze and how different each match can be as their reason for their positive score. For the negative views, the difficult in traversing the maze was a common comment. This difficulty in traversal was mostly seen in the T-junctions with one participant stating that “the multiple paths made it hard to go through as you had to stop.” This issue was mention previously during my analysis of the current system earlier on.

How good was The Amazing Maze Game’s control scheme? + Why did you/did you not enjoy The Amazing Maze Game’s control scheme?

Here the results were generally negative, with The Amazing Maze Game achieving an average score of […]. This poor score was received due to the “slow response” of the buttons and the “difficulty in going through the maze”. A few people commented that if it weren’t for the slow character movement they would have “enjoyed the game considerably”. There isn’t really a lot to elaborate on this result as the issue of controls in The Amazing Maze Game were known to me well ahead of the questionnaire. This issue was commented earlier on during my analysis of the current system.

How fast-paced was The Amazing Maze Game?

The Amazing Maze Game received an average score of […] for this section. The game was generally seen as slow due to the difficulty participants had in traversing through the T-junctions of the maze. This slow pace wasn’t inherently bad, however, as many of the participants stated that the game “was still fun” in spite of the slow controls. This slow pace, however, must be rectified in my final product as my end-user requirements specifically states “a product that is fast paced and requires quick reaction time.” This slow pace can be rectified by using a script that specifically allows the user to stop at T-junctions, as mentioned before.

**Evaluation of “Questionnaire – The Amazing Maze Game”**

Overall, the reception of “The Amazing Maze Game” was generally negative as a result of the poor movement mechanics and poor aesthetic the game had to offer. This negative reception was compounded with the fact that participants had played Tomb of the Mask just prior, which generally had faster movement and better aesthetics than The Amazing Maze Game.

Overall, the negative reception from The Amazing Maze Game has shown the potential limitations of my product, and what issues must avoid to ensure that I correctly align with the needs of my end-users.

The first issue with The Amazing Maze Game was with how the controls functioned around corners. I’ve already mentioned this previously during the analysis of the current system, but the questionnaire just goes to show how important implementing a solution to this.

Another issue with The Amazing Maze Game according to the participants was how the movement was manually and had a slow response. This issue mostly helped feed into the other issue of cornering. This slow manual response from the movement controls can be fixed by using something similar to the Tomb of the Mask, where a single press/slide results in the character moving in the respective direction indefinitely.

**List of Key Points from “Questionnaire – The Amazing Maze Game?”**

* Variability of the maze is important for participants.
* Short mazes are more enjoyable than long mazes for general audiences.
* T-junction solution has to be completed successfully to ensure users can go at a face pace

**Interview Transcript**

The interview transcript below is with a person from The Langley Academy who I had asked to play Tomb of the Mask and The Amazing Maze game. They played each game for 5 minutes prior to the interview. The user was a seventeen year old male named Hafiidh Mrehe.

Interview with Hafiidh Mrehe

Me: What feature did you like in each game?

Hafiidh Mrehe: In Tomb of the Mask, I enjoyed how fast it felt and the controls were really responsive. The game had a nice sprite art style which added to the flavour of the game. The sprites had this sort of retro design thing going for it which made it pop a little more in comparison to other app games I’ve played.

The Amazing Maze Game had a lot of features; the local multiplayer, AI and stuff. I enjoyed playing with them for a bit but they became a bit repetitive after a while. I think this boredom mostly came from the annoying controls though, and less about the game itself.

Me: What feature did you dislike in each game?

Hafiidh Mrehe: Tomb of the Mask didn’t really have anything that I did not enjoy. I think the game could get boring after a while as there really isn’t any competition or local interaction or anything, but I did still enjoy it.

The Amazing Maze Game had a lot of features that I didn’t like personally. I think the premise is there and I think the game has a lot more features than Tomb of the Mask, but it’s the control scheme though that lets it down I think. The maze game had a super slow character speed which made it tedious to go through. I think the idea of a maze game is cool but I think the maze has to be a bit smaller to make it more appetising for the general person.

Me: What new feature would be the most important to you?

Hafiidh Mrehe: I think a Tomb of the Mask with more features like The Amazing Maze Game would be great. Being able to play Tomb of the Mask or some other game with local multiplayer features would be amazing.

**Evaluation of Interview Transcript**

The interview done with Hafiidh Mrehe mostly aligns with the opinions presented in the previous questionnaires. This is reassuring as it demonstrates that the majority of my end-users have similar opinions when it comes to the direction of my proposed solution. The interview, however, has clarified some aspects that I need to work on.   
  
The interview clarified whether a unicursal or multicursal approach was preferred. Mrehe stated that the multicursal maze design was preferred as the variation in pathway would make the game a lot more interesting competitively. This makes a multicursal maze design the better alternative as it clearly aligns with my end-user requirement for a competitive game, having support from both the interview transcript and the general questionnaires.

The interview also clarified whether users were interested in the implementation of multiplayer functionality in the program. I mistakenly didn’t include this question in my previous questionnaires, but I was able to ask Mrehe for his opinions on multiplayer. He felt that multiplayer would be a good inclusion for the product, which is hopefully representative of other users in the questionnaire.

### Analysis – Limitations

Cornering

The issue of cornering was a common concern for many of the participants in the questionnaires, particularly for The Amazing Maze Game. This issue should be fixed easily through the implementation of the T-junction script I hypothesised. The only potential issue with my proposed solution for cornering, however, is that the speed may be compromised for better controls.

Multiplayer Control Scheme

One of the issues with multiplayer on a mobile game is that the screen can’t be shared simultaneously between two people, as the inputs between the two players would get mixed up, making character movement difficult. This issue is a natural limitation with the hardware and I’ll have to adapt the project accordingly.

This issue can be fixed via the use of a PC rather than a mobile. A PC would work well with multiplayer as there are multiple keyboard inputs available for each player. Player 1 could use WASD keys whilst Player 2 could use the arrow keys. This solution does, however, limit the amount of people that can play my game, and it also removes the portability of the product. Thus, I still want to implement a mobile approach for my product.

To implement a multiplayer functionality to my program, whilst also keeping the portability, I must find an approach that allows the players to connect from individual screens. This could be done by using a peer-to-peer system that allows local mobile players to connect to each other. I have little idea as to how to implement this, but it should be feasible in the C# language.

C#

One of the natural limitations with C# is that there aren’t any graph data structures natively in the program that I have come across. Implementation of graph data structures in external libraries are few and far between and the implementation of these graph data structures differ to what I want in particular. I’ll have to create my own class that replicates a graph data structure to correctly implement my CPU algorithm.

Another limitation with C# is that it cannot use global variables due to it being OOP. This makes it difficult to modify variables in different scripts.

Hardware

Another limitation that would be difficult to work around would be the users’ own hardware limitations. The product that I’m creating isn’t hardware intensive, but the Unity Engine itself has system requirements. This means that a portion of my potential user base won’t be able to play the game as their device would fit the necessary system requirements.

### Analysis – Success Criteria

**Primary Objectives**

Below are the primary objectives.

Basic Feature Requirements

1. The user must be able to compete against an opponent in the maze (computer-generated or human)
2. The game must end if the player or opponent reaches the end-point of the maze.
3. The user must be able to control their character towards the end-point of the maze.
4. Each maze layout must be randomly generated.

Design Requirements

1. The game must have an arcade aesthetic (8-bit sprite, music, simple gameplay, etc.)
2. The screen size will be 1920×1080

Processing Requirements

1. It must be determined with each movement whether the character collides with the wall of the maze, in which case the character must stop.
2. It must be determine with each movement whether the character collides with the end point, in which case game should end and determine the winner.

Input Requirements

The user must enter either an arrow key (or the WASD keys) to determine the movement of their character.  
The user must click on certain icons (BACK, QUIT, PLAY, etc.) to navigate through menus.

Output Requirements

The position of the player’s character must be visible in real-time.  
The position of the computer-generated opponent’s character must be visible in real-time.

**Secondary Objectives**

The objectives below are not necessary and are instead desired objectives that would aid in satisfying my end-user. If I notice that these requirements may hinder the development of the game in some way, I’ll be sure to remove them.   
  
Design Requirements  
The colour of the character must be optional for the user.

Input Requirements  
The user will have an optional touch screen input device.   
  
Output Requirements  
The current level number must be displayed.

Hardware/Software Requirements

The system will be written in Unity, and so as a guide the hardware that runs the game should be similar to those necessary to run Unity. I have obtained the requirements to run Unity from the official Unity site

## Design

### Design - Problem Decomposition

**Main Menu Procedure**C:\Users\nuno.dias-goncalves\Downloads\Untitled Diagram (1).png

During the design phase, I decided that my main menu would have a “Standard Game” option for those that want to play against the CPU and a “Multiplayer Game” option for those that want to play against another person locally. I decided to split these two game options into separate procedures right in the beginning as I wanted the user to be immediately aware of what game mode they’re playing in, to prevent confusion. The two other options: “Settings” and “Exit” would be for those that want to adjust the sound or lighting of the game or leave the game respectively. These options were split again for the same reason as above, to prevent confusion.

All procedures (except Exit) will loop back to the main menu after they are done executing (e.g. once the standard game is over, the main menu will appear again).

**Standard Game Procedure**

C:\Users\nuno.dias-goncalves\Downloads\Untitled Diagram2.png

The standard game procedure is mainly dependent on the user input and who reaches the end-point first. The CPU object will be controlled by using the depth-first sorting algorithm. The difficult of the CPU will increase by an increase in the value speed.

For a mobile phone version, this standard game procedure could be easily implemented by changing the control scheme to detect touch screen inputs. This would make it similar to Tomb of the Mask, which had a positive reception with my end-users.

**Multiplayer Game Procedure**

C:\Users\nuno.dias-goncalves\Downloads\Untitled Diagram2.png

The multiplayer game procedure is similar to the standard game procedure.

For a mobile version, a multiplayer game procedure would be difficult to implement as the touch screen inputs cannot be shared. Therefore, a peer-to-peer system would have to be implemented between mobile phones to play the game, potentially making development more difficult overall.

**Design – Structure of Solution**

To determine the structure of the solution in my project, I decided to create design prototypes of what my game could potentially look like. I then later asked a handful of volunteers from my desired demographic (who had previously done the questionnaires) to look at the prototypes and criticise them.

Main Menu Design – “PC”

MENU DESIGN ONE: This initial menu design was created without any initial user feedback (apart from the results of the questionnaire). In this design I tried to replicate the style of old arcade game layouts (e.g. Galaga and Pac-man) by implementing a pure black background with white text and arrows. The arrows would move discretely between the options when pressing the WASD keys or the arrow keys. I also decided to add a grey streak behind the background as a stylistic contrast to the completely black background. This main menu design was created to look as simple as possible, in order to best align with the results of the questionnaire.

The feedback for MENU DESIGN ONE was mixed. Some of the volunteers liked the minimalistic design of the menu whilst others thought the menu was too simple and lacking in colour. I decided to create another prototype with a bit more colour to cater to those that disliked this menu design and to see whether that menu would get a better reception overall.   
  
MENU DESIGN TWO: This game design was created using some of the feedback from the previous maze design. I opted to include a red background as the colour red is commonly associated with dynamism and activity more so than other colour. Since I want my game to be competitive and fast paced, I decided to use this colour (it also provides a contrast to the black background). The menu options were placed further towards the bottom left hand corner to provide a visual dynamic angle to the design.

The feedback for MENU DESIGN TWO was a lot more positive than the previous menu. The addition of red seemed to make the menu better aesthetically according to my clients. I decided to use this menu design due to the positive reception it garnered.

Game Design – “PC”

GAME DESIGN ONE: This initial game design was created without any initial user feedback (apart from the results of the questionnaire). In this design I opted to remove menus or instructions. I decided to remove these features as I wanted the user to learn the controls of the game without having to shove an instruction screen towards them. The product I’m creating requires focus in order to beat your opponent, and the use of instruction screens or pause buttons in the game screen would, more often than not, break that focus/immersion in the game, This style of game design may lead to more initial confusion for the audience, but through the implementation of affordances (clues about how an object/product should be used) the audience should be able to learn the game naturally just by playing.   
  
Unfortunately, in the user feedback, the response was mostly negative. The volunteers found the lack of instructions and menus to be too annoying for them. One volunteer stated that they’d be “confused” about how to play the game. This criticism can be resolved by implementing a pause menu into the game screen design to make it a bit easier to understand. I could also implement a semi-transparent picture of the WASD keys to appear for a while so that the user understands what buttons they are meant to press. I still want to implement affordances into my game design so that the user can naturally understand the game without having to go through a wall of instructions however.

GAME DESIGN TWO: In this design I took the user feedback into account to create an overall better looking game design. I decided to add a BACK option so that the user can go back to the main menu during the game. I also decided to add a black bar behind the BACK option to act as a visual contrast and make it easier for the user to identify the BACK option whilst they play.

Volunteers were overall more positive with this game screen design than the previous one. Some still commented on the game being too simple for their liking but I decided to use GAME DESIGN TWO as it’d be easier for me to program whilst also aligning with the arcade aesthetic.

Main Menu Design –“Mobile”

MENU DESIGN ONE: This mobile design for the game was created with no initial feedback (apart from the questionnaire done in Analysis). This menu design was created to be a simple as possible to prevent any potential touch errors. The buttons were made to be large so that the user can clearly identify and touch the button they want, and the background was made to be dark so that the user can clearly identify the differences between the buttons. I wanted the menu to be as simple as possible to aid the arcade aesthetic, and to declutter the small mobile screen the user would have.

Surprisingly, there were very little criticisms with this design. Previously, my menu design for the PC was deemed too simple, but the mobile design was seen as more suitable by the participants due to the smaller screen. One criticism however, was that buttons cause

Game Design – “Mobile”

GAME DESIGN ONE: This game design was created with no initial feedback again (apart from the questionnaire done in Analysis). This game design, similarly to the menu design, focused on simplicity more so than ease of use. The game screen was designed to be a simple as possible, to prevent potential issues with the small touch screen. I also tried to implement affordances into my game design once again.

The feedback for this menu design was mostly positive, with a few negative comments. Most users were okay with the simple design as the game would be on a mobile device. The issue, however, for a few of the participants was that the red background may hurt their eyes, especially during the night when the eyes are accustomed to dark screens. Most modern apps have a designated night mode for these sorts of issues. Due to this potential problem, I decided to change the colour of the background to one that’s more appropriate and easier for the eyes.

GAME DESIGN TWO:

Design – Algorithms  
  
**Multicursal Mazes Algorithm**

For my project, I’ll be using the depth first search (DFS) with recursive backtracking algorithm to create my maze. While there are other algorithms to create mazes, DFS is the simplest for me to program personally and it’s an algorithm that is often regarded as the easiest algorithm for other programmers to understand.

As explained on Wikipedia, this is the recursive backtracking maze generation algorithm:

|  |
| --- |
| Make the initial cell the current cell and mark it as visited                  WHILE there are unvisited cells                                  IF the current cell has any neighbours which have not been visited                                                  Choose randomly one of the unvisited neighbours                                                  Push the current cell to the stack                                                  Remove the wall between the current cell and the chosen cell                                                  Make the chosen cell the current cell and mark it as visited                                 ELSE IF stack is not empty                                                  Pop a cell from the stack                                                  Make it the current cell |

This is the algorithm that will base the majority of my program on. Most of my program will be making this work in the C# language so I can run it through an empty object in Unity. With my program, each individual cell will be made through the use of a predefined prefab. Below I made a more detailed expression of the algorithm using pseudocode.

|  |
| --- |
| Remove [current\_cell] from [unvisited\_list] (i.e. mark as visited) WHILE no.[unvisited\_list] is greater than zero then                 IF [current\_cell] \*has unvisited\_neighbours greater\* than zero then                                 Get [random\_neighbour\_cell] from  [neighbour\_list]                                 Add [current\_cell] to stack                                 \*Remove the wall between [current\_cell] and [random\_neighbour\_cell]\*                                 Let [current\_cell] equal [random\_neighbour\_cell]                                 Remove [current\_cell] from unvisited list (i.e. mark as visited)          ELSE IF stack is greater than zero then                                 Let [current\_cell] equal the most recently added cell in the stack                                 Remove [current\_cell] from the stack. |

 This pseudocode will make up a part of my program. The highlighted lines above are areas that will act as functions in the program as there isn’t any equivalent function in C#. Below I have written two pseudocode scripts for these lines.

|  |
| --- |
| IF neighbour\_cell has a greater x coordinate than current\_cell then                 Remove right\_wall of current\_cell                 Remove left\_wall of neighbour\_cell ELSE IF neighbour\_cell has a lower x coordinate than current\_cell then                 Remove left\_wall of current\_cell                 Remove right\_wall of neighbour\_cell ELSE IF neighbour\_cell has a greater y coordinate than current\_cell then                 Remove top\_wall of current\_cell                 Remove bottom\_wall of neighbour\_cell ELSE IF neighbour\_cell has a lower y coordinate than current\_cell then                 Remove bottom\_wall of current\_cell                 Remove top\_wall of current\_cell } |

|  |
| --- |
| Get\_Unvisited\_Neighbours(current\_cell)  { Create a new list called [neighbours]  Let new variable [neighbour\_cell] equal [current\_cell] Store [current\_cell] position in a new vector called [current\_position]                 FOREACH vector in neighbourPositions                                 if neighbour\_cell is in the desired maze layout then                                                 neighbour\_cell equals allCells[neighbour\_position]                                 if neighbour\_cell is in unvisited\_list] then                                                 add [neighbour\_cell] to list [neighbours] } |

(The algorithms above should remain fundamentally the same for all other programming languages).

Another thing I need to take into account when creating the game is the grid which the maze will sit on. A modifiable grid with individual cells is necessary for the recursive backtracking algorithm to work. The grid must be created beforehand so that the Maze\_Generator function can modify it accordingly. The maze generator will influence each cell it passes through within the grid. Because the grid will be made of prefabs each cell will have to undergo the process of being referenced and instantiated as a new game object. During development, I’ll set up this reference and instantiation as a function as it’ll be repeated for each grid cell in the game.

Below is my proposed Create\_Maze\_Layout() function.

|  |
| --- |
| Create\_Maze\_Layout {                 FOR EACH row in maze row                                 FOR EACH column in maze column                                                 CREATE cell                                                 INCREASE y.position by cell size                                 y.position equals 0                                 INCREASE x.position by cell size } |

So here are the fundamental steps of my program as of right now for the multi-path maze design in my maze object.

Each component of the wall (such as colliders and sprite) will be set up beforehand in the prefab.

**Unicursal Maze Algorithm**

To create a unicursal maze I require a different set of algorithms to account for the fact that dead-ends cannot be created. Researching online I have come across one algorithm that could help me develop a single path maze; the space-filling curve. The space-filling curve is an algorithm that designs […]. The lack of a true single-path maze generation algorithms makes it difficult to develop unicursal mazes, as the lack of an algorithm means that I’ll have to research or “brute-force” the algorithm on my own accord. Due to this issue and due to the severe compromise a unicursal maze would have on the complexity I will instead use the other solution I came up – where I create a character script that stops movement at every T-junction in a multicursal maze.

**T-junction Approach Algorithm**

For the T-junction approach I need to take a list of all the cells that have neighbours equal to three. This list can be created after the maze generation algorithm. To identify the T-junctions, I’ll make a nested for loop to individually identify each cell of the grid, and check whether their wall IDs have been deactivated. If their wall ID has been deactivated more than three times, then it must be a T junction cell.

Once the T-junction list is created I need to implement a vertical and horizontal pass-through platform in the cell. This pass-through platform will stop the user at the first initial collision but allow the character to go through the collider once that press the appropriate directional key again.

Below is my proposed pseudocode for the T-junction approach.

Currently, I have no approach in implementing a pass-through wall into the T-junction cells. I’ll need to do further research to learn how to do this.

**CPU Algorithm**

For the CPU algorithm I must either create a script that allows an object to progressively go through the maze (this could be done using DFS) or use a pre-made script that uses general path finding. For this algorithm, I’ve decide to opt for pre-made script.

I decided to opt for pre-made scripts as I had trouble with the pseudocode when creating the bespoke CPU script. My initial idea in designing the CPU script was to use waypoints assigned to each vertex in the maze. Each waypoint would link to another waypoint, creating a railroad system for the CPU to travel to the exit. One of the difficulties in creating this script however was writing C# pseudocode that used the graph data structure correctly. I was unable to find or create a data structure class that successfully incorporated the graph structure whilst also working for the specific conditions of my maze, thus I decided to find a pre-defined script. Below was a portion of my original C# script for the bespoke CPU. Below is a portion of my initial pseudocode for the CPU.

   public class Node {  
                public Vector2 position;  
                public List<Node> neighbours;  
                public Cell cell;  
   }

private void Traversal() {  
                List<Node> topnodes = null;  
                int count = 0;  
                for (int y = 1; y <= maze\_rows; y++) {  
                                bool prv = false;  
                                bool cur = false;  
                                bool nxt = false;  
                                current\_cell = allcell\_dictionary[new Vector2(1, y)];                            
                                current\_cell.cScript.GetComponent<SpriteRenderer>().color = Color.blue;  
                if (allvertices.Contains(current\_cell)) {  
                                                cur = true;  
                                }

                                Cell nxt\_cell = allcell\_dictionary[new Vector2(1, y)];  
                 if (allvertices.Contains(nxt\_cell))  {   
                                                nxt = true;  
                                }

                   Node leftnode = null;

            for (int x = 1; x <= maze\_columns; x++) {  
                prv = cur;  
                cur = nxt;  
                nxt\_cell = allcell\_dictionary[new Vector2(x + 1, y)];

                if (allvertices.Contains(nxt\_cell)) { nxt = true; }  
                else { nxt = false; }

                Node n = null;

                if (cur == false) { continue; }

During development, I’ll try to implement my own approach for the CPU through brute force, but if I’m unable to create the CPU successfully through brute force I’ll use a pre-made path finding script.

**Multiplayer**

**One of the limitations I mentioned previously was implementing a local multiplayer functionality into a mobile game due to the screen’s inability to differentiate between multiple inputs. To counter this issue, I’ll have to implement a peer-to-peer system or another multiplayer system into my product that allows user to connect to each other locally and play on separate mobile screens. Otherwise, I’ll have to scrap the multiplayer functionality of the mobile game altogether and keep that feature exclusive to PC.**

**Researching online, I have noticed that P2P networking is often inappropriate for multiplayer games as it wasn’t specifically designed for multiplayer integration. It was designed for file sharing. The unity documentation has advice on setting up a multiplayer project, listing the necessary features of a network game. Below is the list of necessary features for the multiplayer project.**

* **Network Manager**
* **User Interface**
* **Player Prefabs**
* **Scripts and GameObjects which are multiplayer aware.**

**Design – Data Structure**  
  
Below I have made a list of key variables, data structures and classes that I predict will be in my program. In the program a new class must be made from the prefab to create each instance of a cell.

MazeGenerator Script

|  |  |  |  |
| --- | --- | --- | --- |
| Variable Name | Class | Data Type | Purpose |
| maze\_rows | MazeGenerator | Integer | To determine how many cells the maze will have horizontally. |
| maze\_columns | MazeGenerator | Integer | To determine how many cells the maze will have vertically. |
| Allvertices | MazeGenerator | List<Cell> | To hold all the vertices for the Depth-First search algorithm (optional) Creates the CPU. |
| unvisited\_list | MazeGenerator | List<Cell> | To hold all unvisited cells for the maze generation algorithm. Creates the actual maze. |
| tjunction\_list | MazeGenerator | List<Cell> | To hold all T-junction cell for the T-junction algorithm to use later on. Aids in controls for players. |
| cell\_Prefab | MazeGenerator | GameObject | To act as an object that can be instantiated. |
| current\_cell | MazeGenerator | Cell | To act as the current cell from which neighbours will be checked against from. (The data type Cell will have to be created from a new class) |
| Maze | MazeGenerator | GameObject | To present the maze in the in-scene view. |
| Stack | MazeGenerator | Stack | To hold the items received from the maze algorithm. Creates a list from which visited cells be checked as they approach a dead-end. |
| neighbour\_vector | MazeGenerator | Vector2 | List of potential vectors the neighbour cell could take. Easier to use as a list as it can be used in tandem with a for loop. |
| cell\_size | MazeGenerator | float | To take the local cell size. Helps determine how much each cell will be offset by. |
| start\_position | Maze\_Generator | Vector2 | The vector start position of the cell in the 2D plane. |
| spawn\_position | Maze\_Generator | Vector2 | The spawn position of the current cell in the nested for loop. Each position will go up by half a cell at each iteration of the loop. |
| GetNo\_ofAllNeighbours | MazeGenerator | Integer | A variable created from a function that returns the no of all neighbours in a current cell. An integer value. |

CellScript

|  |  |  |  |
| --- | --- | --- | --- |
| Variable Name | Class | Data Type | Purpose |
| Wall | CellScript | GameObject | To hold the wallL component. Enables the activation and deactivation of the wallL child object. |
| wallR | CellScript | GameObject | To hold the wallRcomponent. Enables the activation and deactivation of the wallR child object. |
| wallU | CellScript | GameObject | To hold the wallU component. Enables the activation and deactivation of the wallU child object. |
| wallD | CellScript | GameObject | To hold the wallD component. Enables the activation and deactivation of the wallD child object. |
| trigger | CellScript | GameObject | To hold the triggerL component. Enables the activation and deactivation of the triggerL child object. |
| trigger | CellScript | GameObject | To hold the triggerR component. Enables the activation and deactivation of the triggerR child object. |
| triggerU | CellScript | GameObject | To hold the triggerU component. Enables the activation and deactivation of the triggerU child object. |
| triggerD | CellScript | GameObject | To hold the triggerD component. Enables the activation and deactivation of the triggerD child object. |

TriggerScript

|  |  |  |  |
| --- | --- | --- | --- |
| Variable Name | Class | Data Type | Purpose |
| grandparent\_name | TriggerScript | string | To obtain the name of the grandparent for the current item. Helps determine whether the cell is in the maze for player one or for player two. |
| parent\_name | TriggerScript | string | To obtain the name of the parent of the current item. Helps determine which cell the current trigger wall is a part of. |
| opp\_wall | TriggerScript | GameObject | The actual GameObject of the opposing wall to the current item. |

CPU Script

|  |  |  |  |
| --- | --- | --- | --- |
| Variable Name | Class | Data Type | Purpose |
| prev\_cell | CPU | Cell | To |
| currentCell | CPU | Cell |  |
| Nodevertex | CPU | bool | To determine whether the current cell is a vertex of the maze or not. |
| jj |  |  |  |

Design – Test Strategy

For my testing strategy, I’ll first use white-box testing (structural testing) to test the internal structure of the program. I will then use black-box testing (functional testing) to test the functionality of the program, going by the requirements of my end-user. Finally, I’ll use beta testing with my volunteers in order to expose the product to real use and detect problems and errors I may not have anticipated.

The table below will act as a basic template for white box testing. The following table is designed to check the actual outcome against the expected outcome of the test. The input data works by giving a typical data (expected), erroneous data (data that would cause the system to thrown an exception, e.g. incorrect data type) and extreme data (less expected data, e.g. Blank fields).

| Test No. | Description | TEX (Typical, Erroneous, Extreme Input) | Expected Outcome | Actual Outcome | Corrective Actions |
| --- | --- | --- | --- | --- | --- |
| 1. | Selecting a menu option. (Does the selection with the menu option work?) | T: A menu option has been selected appropriately using the arrow navigation system. | The screen will transition to the appropriate scene. |  |  |
| E: No menu option has been selected. | No scene transition will occur. |
|  | Fading in and out transition. (Does the game transition between the scenes with the correct animation?) | N/A | The fading in and fading out animation will be called as appropriate whenever the user moves between scenes. |  |  |
| 2 | Modifying the settings. (Am I able to modify the settings of the game?) | T: A setting option has been modified appropriately. | The setting will be modified appropriately, and the setting changes will translate to the game scene. |  |  |
| E: No setting option has been modified by the user. | No settings will be modified by the user. |
| 3 | Moving the character object in PC. (Can I move the character quickly through the maze using WASD keys?) | T: The character object is moved using the appropriate directional character keys. | The character will move in the assigned direction and slide along the maze. The character object will stop when it collides with a cell wall. |  |  |
| E: No appropriate directional character keys are used to move. | The character object will not move. |
| 4 | Moving the character object in mobile. (Can I move the character quickly through the maze via swiping?) | T: The character object is moved using the appropriate swiping mechanics on the mobile phone. | The character will move in the assigned direction and slide along the maze. The character object will stop when it collides with a cell wall. |  |  |
| E: No appropriate swiping mechanics are used to move. | The character object will not move. |
| 5 | Moving both character objects simultaneously (multiplayer). (Do the movements interfere with one another?) | T: Both character objects are moved using the appropriate directional character keys. | Both character objects will move simultaneously along their own mazes. There should be no overlap between the control schemes. |  |  |
| E: No appropriate directional character keys are used. | Each players’ respective character object will not move. |
| 6 | Player collision with the end\_cell during the standard game scene. (Do I transition to a different maze?) | T: The character object collides with the end\_cell directly | The scene will transition to the next level after it completes the collision. |  |  |
| E: The character object attempts to collide with the end\_cell through one of the thin cell walls. | The scene will not transition as the character object did not collide with the end\_cell directly. |
| 7 | CPU collision with the end\_cell during the standard game scene. (Does the game end successfully?) | N/A | The scene will declare the CPU as the winner and transition back to the main menu scene. |  |  |
| 8 | Player collision with the end\_cell during the multiplayer game scene. (Does the game end after a player has collided with the end\_cell?) | T: One of the player’s character object collides with the end\_cell. | The scene will declare that player as the winner and transition back to the main menu. |  |  |
| E: One of the player’s character object attempts to collide with the end\_cell through one of thin cell walls. | The scene will not transition as the character object did not collide with the end\_cell directly. |
| X: Both player’s collide with the end\_cell simultaneously. | The scene will declare a random player as the winner and transition back to the main menu. |
| 9 | Does the character stop at the edge of every wall?  Does the character object spawn at the beginning of every maze? |  |  |  |  |

##### **Development**

**Purpose – Maze Generator Script**The purpose of the maze generation script is to procedurally generate a maze layout that a character object can successfully traverse. The maze generation script requires a prefab to instantiate each cell in the maze, and it requires the user to input how many maze rows or maze columns they want in the maze object.   
  
**Brief Development Description (e.g. how it went) – Maze Generator Script**The prototype of the maze generation script was created immediately after the design portion of my project was completed. The prototype works by using the recursive backtracking pseudocode that I created back in the Design portion of this document. The prototype had some initial failures as I was creating it (such as general syntax errors or null reference exception errors). These initial failures were caused by my lack of knowledge on C# language. I was, however, able to develop the maze generator script after learning the basics of the C# language and by closely following the pseudocode that I had described. Below is a prototype of the maze generation script that I created.   
  
**Prototype Description – Maze Generator Script**   
  
(1) At the beginning of the Maze Generation Script, I initialized the variables that I needed for the entirety of the program. I also created the cell class here for later use during the generation of the grid layout.

(1) The cell class requires several attributes to work successfully. To activate and deactivate the walls of the cell class successfully, I made sure to create a bespoke script for each cell. In that script, the left wall, right wall, up wall, and down wall are all attached as well as the trigger walls. Through this script, I am able to successfully deactivate and activate walls as appropriate for the maze layout.  
  
(1) I had initially decided to serialize the maze\_rows and maze\_columns variables alongside the cell\_prefab and the maze\_parent as I wanted to easily change the size of the mazes in the Unity Inspector. I decided to remove the serialization however as there was little need to change the integer value of the row and columns of the maze without changing the location of the maze objects in the script. The cell\_prefab and maze\_parent have to be serialised so that a prefab can be assigned in the scene view and so that the Unity engine can reconstruct elements of the program for later use.

(2) Here I decided to separate the function Create\_Maze() into a discrete simple function to make it easier to understand for other programmers. It also allows the program to be called easily from other scripts if I wish to do so.  
  
(2) The starting\_position and cell\_size variables were calculated by either using the localscale method to determine the size of the prefab or by simple arithmetic for vectors. This section of the code remains largely the same in the final solution. I did, however, clean some of the variables up that I noticed I did not use in the final solution (e.g. I created the allvertices list here after an initial attempt at the CPU script).  
  
(2) As stated in Design, a nested for loop was an essential component in developing the grid layout of the maze. Here, I called the Construct\_Cell function (a function created later in the program) to instantiate each cell. I ensured that the spawn\_position of each cell was incremented by cell size \* 2 so that the cell is instantiated in the right position with respect to the next cell. This section of the program remains largely unchanged in the final solution.

(2) After the grid layout is produced, the order of execution for the remaining functions was goes as follows: Run\_BackTrackingAlgorithm, MakeExit, RunTJunctionAlgorithm.   
  
(3) The RunBackTrackingAlgorithm() function works identically to the procedure mentioned in the Design portion of the document. The function uses certain bespoke functions (such as GetNoofNeighbours() or GetNeighbourList()) as there were not any equivalent functions in C# that functioned as I wanted them to. This function remains largely the same in the final solution, albeit with minor changes (e.g. changes to the variable names, changes to which functions are being used for the procedures, etc.)

(3) The Run\_TJunctionAlgorithm() function works via the use of GetNo\_of\_Neighbours() function to determine whether the current cell is a T-Junction. I described the Run\_TJunctionAlgorithm() function more in the prototype description of trigger script, as the trigger script works directly with this function.   
  
(4) The MakeExit() function works by retrieving a list of cells that lie on the top edge, taking a random cell and then removing that random cell’s top edge to create an exit. To differentiate the end\_cell, I modified the colour of the Sprite Renderer for the end\_cell to make sure the user can easily identify the location of the exit.

(4) The MakeExit() function, after finding the end\_cell, assigns the end\_Cell with the BoxCollider2D() component as well as a script that detects collisions. The script functions differently depending on what game mode the user is playing in (e.g. the game would increment the CPU by one in the Standard\_Game\_Scene but not in the Multiplayer\_Game\_Scene). This allows the current scene to transition appropriately to the next level once either the CPU or a player has collided with the end\_cell of their own respective mazes.

(4) In one of the earlier prototypes, I had created the MakeExit() function to take all potential edge cells of the maze (e.g. the entire left edge of the maze would be included in the list of edge cells). This improved the variability of the maze locations as the exit could appear on the left side or the right side. I removed this feature in the final solution, however, as I noticed that the mazes were very easy to solve as the maze exit was generally closer to the character object.

(5) The GetNoOfNeighbours() function works by checking which wall IDs are inactive for a particular cell (e.g. cells that are T-Junctions should have three walls IDs that are deactivated). In one of the earlier prototypes, I had used an IF statement to check for the activation of wall IDs and to check whether the neighbour cell existed in the game scene. After some initial background testing, however, I realised that checking whether the neighbour cell exists was not necessary to determine whether the current cell was a T junction. I decided to remove this functionality in the final solution.

(6) The GetUnvisitedNeighbours() function works by determining which cell is a neighbour and using an IF statement to determine whether that neighbour cell is present in the unvisited list and to also determine whether that neighbour exists as a game object in the game scene. It works similarly to the GetUnvisitedNeighbours() function.

(6) The GetListOfNeighbours() function works by check which walls are activated and deactivated for a particular cell and adding that to a string list that denotes which neighbours a cell has. This function also works similarly to the previous two functions mentioned. During testing, I’ll check to see whether all these functions are necessary and attempt to reduce their length.

(7) The RemoveWall() function deactivates the appropriate wall of a cell by setting the corresponding cScript to false. The RemoveWall() function uses a wall ID to determine which wall it should deactivate. This makes the process of deactivating walls faster.   
  
(7) The DeleteSimilarWall() function deletes the walls of two corresponding cells that sit next to each other in the game scene. The function figures out the direction of the neighbour given the current\_cell by checking the y and x of the grid position. The function then deactivates the corresponding walls appropriately. The DeleteSimilarWall() function works in tandem with the RemoveWall() function to complete its function. The prototype version of this function remains largely the same in the final solution.

(8) The ConstructCell() function instantiates and sets the transform variables of the cell object by using the Vector2 pos to determine the transform of the cell whilst using the keyPos to position of the cell in relation to the actual grid. The name of the cell is given by the keyPos x and y. Other variables, such as the neighbours list or the trigger script are set a default variable.  
  
The development of the maze generation script was done to ensure the variability of each maze. The maze (being randomly generated) ensures that the product is always different for the user, preventing the potential boredom that would come when going through the same maze. During analysis I commented on this variability during “Justification of Computational Solution”. This maze generation script successfully accomplishes this variability via the backtracking algorithm.

**Review – Maze Generator Script**

The maze generator script seems to work efficiently, but there are several current problems with the implementation that have to be addressed in the final solution.

The first main issue with the maze generation script is that multiple functions share similarities to one another. This makes the script inefficient as lines of code are used to store essentially the same base algorithm. To fix this inefficiency, I must identify functions that share common features with one another and try to merge them together.

The second issue with the maze generation script is that certain variables and comments are not clearly defined in the script. This lack of comments and appropriate variable names would make it difficult for other programmers to successfully understand the implementation of my project. In the final solution, I must modify my variables appropriately, and add comments to ensure that my script is easy to understand for others that may wish to use the code.

**Final Solution - Maze Generation Script**

\*- In the final solution for the maze generation script, I made sure to include as many useful comments into the code as possible to ensure that other programmers could understand the reasons why I did a particular action in my code. In addition to comments, I also decided to rename the majority of my variables into more appropriate names. This was done so that other programmers could understand what my variables mean and what influence they have on the code later on. The line indentations were aligned correctly through the use of a program.

\* In the final solution, I decided to remove the unnecessary lines that were in my program. I did some research to determine whether any algorithms/functions could be simplified in the program, and simplified them accordingly. For example, I simplified the IF statement of the CheckNoOfNeighbours() function and removed the check for whether the neighbours existed or not. I also identified functions that shared similarities to one another and removed them as appropriate to improve the efficiency of my program.

**Prototype/Development of the T-Junction Script**

**Purpose – T-Junction Script**

The purpose of the T-Junction Script is to ensure that the player can successfully traverse the T-Junctions of the maze without having to slow down their speed. The T-Junction script aims to stop the player at every T-Junction they pass so that the user can successfully choose which path they wish to take without having to stop the object themselves. The T-junction script uses the OnColliderEnter() function native to C# in order to detect the player object and stop them accordingly by enabling the collider property of the walls around the cell.

**Brief Development Description (e.g. how it went) – T-Junction Script**  
The prototype of the T-Junction script was created after I completed the development of the basic maze generation script. In Analysis, I stated that I was going to use a pass-through platform that stops the character object at each T junction. I ultimately decided against the use of a pass-through platform, however, as I wasn’t sure how I would implement that pass-through functionality into my program in a 2D plane. Instead, I decided to use a “pressure plate” trigger system where one wall would cause the next wall to act as a collider for a set period of time. This would stop the user at the given position and mimic the conditions of the pass-through platform. This “pressure plate” system also happens to be simple to implement. Below is a section in the maze\_generator script used to detect the T-junctions of the maze.

**Prototype Description – T-Junction Script**

The TriggerScript of each cell works in tandem with the RunTJunctionAlgorithm in the Maze Generator class to successfully implement each script to the appropriate cell. Below is the RunTJunctionAlgorithm, and a description detailing how the function below relates to the trigger script.

The algorithm uses a nested for loop to iteratively go through the constructed maze, and checks the activated walls of each cell. If the program detects a T-Junction (a cell with three or more deactivated walls), the trigger scripts for that cell are activated accordingly.

The script below is attached to every T-Junction that occurs in an instance of a maze.

(1) The ExecuteAfterTime() function delays the execution of the statements inside of the function. The statements below reactivate the wall, allowing the character object to pass through them as normal. The ExecuteAfterTime() function is necessary to prevent users from sliding all the way through the T-Junction as its reactivated.

(1) In the earlier prototype for the TriggerScript, I had used the MeshRender component to change the colour of the cell whenever I detected or modified a T-Junction. This change made the process of checking whether I had detected all T-Junctions easier as I could visually check which cells have been denoted as a T-Junction in the play-view.

(1) The OnTriggerExit2D() is a function that determines whether an object has exited a collision with the object that has the attached script. The OnTriggerExit2D() function is a function native to C# monobehaviour. This function allows me to successfully determine when the player has passed through the T-Junction, which is when the corresponding opposite wall should be activated to stop the player. The wall with the attached script causes the opposite wall to become a collider for a set duration (e.g. left wall causes the right wall to become a collider). This opposing wall stops the character object for a set duration (a duration caused by the ExecuteAfterTime() function mentioned previously). This activation and deactivation of the T-Junction wall object prevents the user from having to thread their character through the gap, as the T-Junction wall object stops the character at the centre of the cell for a set duration of time. The user can then pick their path accordingly.

Ultimately the T junction script was created to solve the issue of cornering in a maze. I previously mentioned this issue of cornering in a maze during the “Evaluation of Current Systems”. I knew that my project would suffer heavily if I were to leave the issue unfixed, as this same issue of cornering heavily influenced the negative reception “The Amazing Maze Game” had in the investigative report. I spent a large portion of my time coming up with potential solutions to solve the problem. This implemented solution seems to work well.

**Review – T-Junction Script**

There are only a few issues that prevent the trigger script from working successfully. The script is able to stop the user at the correct position at the correct moment, and the script is easy to understand for other users. The only criticism

**Final Solution - T =Junction script**

\*After completing the T-junction script, I removed any of the methods that changed the colour of the sprite as the colour of the sprite wasn’t necessary for the user to know. I also made sure to annotate the code for other programmers so that it’s easier to understand.

\* The T-junction script ultimately remains unchanged in the prototype.

**Prototype/Development of CPU Script**

**Purpose – CPU Script**

**The purpose of the CPU script is to create an object that can successfully traverse a maze against the user. The CPU script aims to provide a competitive element for those that want to play the product on their own. The CPU script works by using a bespoke graph data structure that connects all nodes to their respective neighbours in the maze. A maze solving algorithm is then used to determine the path of the CPU. The creation of the graph data structure is done in the maze generation script whilst the BFS algorithm is done in the CPU script.**

**Brief Development Description – CPU Script**

The prototype of the CPU script was created after I had completed the T-junction script and the maze generator script. As I stated in Design, I was not able to create a working pseudo code for the CPU script, making the CPU script difficult to create. For the CPU script I spent a lot of time fixing abrupt issues that appeared in the program and trying to develop a working algorithm to implement. After some initial research, however, I was able to implement the CPU with partial success into my code. To create the CPU script, I decided to create multiple pseudocodes that I could test in C# to see whether they worked or not. Below is my first attempt at the pseudocode.

**Prototype Description – CPU Script**

My first prototype for the CPU pseudocode was created after coming across a video that described maze solving solutions. The video, “Maze Solving – Computerphile” described the process of detecting nodes in a maze given as an image. Michael Pound, a professor from the University of Nottingham, described his approach to the solution carefully. The illustrations of the video showed how Pound’s algorithm worked, and he went on to describe his thought process when he approached the solution. I looked at Pound’s Python code in GitHub and tried to take some inspiration from there. The issue with using Pound’s implementation as inspiration for my pseudo code however was that there were certain functions in his Python code that could not be used for a maze created in C#. His code also differed in that the cell walls were actual, discrete pixels that were colliders instead of just thin walls. I spent a lot of time trying to modify Pound’s implementation to work for my code, but I was ultimately unable to. Because of this, I scraped this pseudocode below.

(1) At the beginning of the pseudocode, I decided to create a class named node. This class has a neighbour list to connect each node to one another successfully. Ideally, the pseudocode would work by recursively going through the maze and creating this class.   
  
(1) Similarly to Pound’s implementation, the start row of the maze would be done separately from the rest of the maze. The start row would check whether the up wall ID is deactivated for the row. If the up wall is deactivated that means that the cell is the exit, thus an instance of the node class must be called.

The second approach was using a pre-defined script from an external source to supplement my code. At this point, I was annoyed with my inability to create the CPU and thus decided to find someone else’s approach to satisfy the need for my end-user. The script seemed to work for a bit but there were several underlying issues with trying to implement […] library into my code. The first issue was that the library was specifically adapted to work in the 3D environment, which meant that several features of the library were unavailable for me to use. The second issue was that the nodes could not be created successfully on the maze object. Similarly to Michael Pound’s code, […] approach to the problem involved taking the walls as discrete nodes that cannot be passed through, making it an unsuitable solution for my program due to its inefficiency.

(1)The pathfinding worked, but it only worked when there were thousands of nodes in the scene.

The third approach for the CPU script was figuring out a solution to the problem on my own. I tried to mimic Michael Pound’s thought process when approaching my own maze, and I thought about what defines a node or a T junction in my maze. I subsequently came up with IF statements that recursively went through each cell of the maze. I drew sketches of mazes and tried to manually go through each of my drawings with the algorithm in mind. The algorithm I came up with was inefficient as some of the cells did not have to be represented by nodes, but it functioned well with my maze implementation so I attempted to produce it in C#.

Below was my first attempt to recreate the CPU script.

(1) I tried to create the graph data structure in the maze generation script. This was done with mild success as I was able to successfully identify each potential node whilst improving the efficiency by removing the horizontal paths. There were, however, notable problems with the implementation.

One of the problems with the above code, however, is that the script could not differentiate between child nodes and parent nodes. The function seemed to work initially (I used the Mesh Renderer component to change the colour of the nodes, which it did correctly), but then I realised during debugging that an infinite while loop between the neighbours occurred here because of the functions inability to differentiate between parent neighbours and child nodes. In a random occasion, the parent neighbour of a node was inserted into the queue for breadth first search (instead of the child neighbour), causing the while loop to occur indefinitely (the parent linked to the child and child linked to the parent). I reduced the size of the maze columns and maze width to see whether the infinite while loop issue continued, which it did. This issue ultimately caused my computer to freeze indefinitely until Unity returned an Out of Memory error.

To fix the issue, I used a visited\_list to denote which nodes have been passed by. I created an empty visited list and used an IF statement to see whether the node in the queue has already been seen. If the node was seen, I made sure to not add it back into the queue. This successfully solved the issue, but this implementation removed the backtracking functionality of the CPU. This removal of the backtracking functionality made it impossible for the CPU to successfully traverse long mazes, but my implementation only uses small mazes making it a sufficient solution.

Below was my second attempt to recreate the CPU script.

\*The issue I had with the above code was that I wasn’t able to transfer some of the variables from the maze generator script to the CPU script. I tried to move some of the variables between the scripts by using the public method, but this method only returned null values in the CPU script. In an attempt to solve this issue I used the keyword static. The use of the static method allowed the variable to be consistent and shared throughout all objects, allowing transfer between scripts. This approach worked, but it made the program inefficient as the NodeCreation() and BreadthFirst() function could not be transferred to the CPU script. Because of this difficulty I tried to find a different approach to this problem.

In the next approach to the CPU script, I tried to make the two scripts (the CPU script and the MazeGenerator script) as singular as possible whilst retaining the same algorithms. Reading online, I found advice from others stating that the keyword “static” should be used sparingly in Unity, as the scripts should be able to function independently of one another. I separated the functions and instead made the CPU script dependent on the actual maze in the play view due to this comment.

Below is my third and final attempt to create the CPU script. I decided to include every line of the script in this section.

(1) The beginning of the code involves the initialization of variables for later use. The variable MoveSpeed was given the static component so that the speed can be increased appropriately whenever the scene was reloaded in another script. This allowed the speed of the CPU to increase for each level whilst resetting the other variables of the script for the maze solving system to work.

(1) In the Start() function, I assigned the CPU with the same Vector2 position as the starting cell of the maze, allowing the CPU to spawn in the correct position. I then executed the function of the script as follows: AllCellDictionaryCreation(), NodeCreation(), and BreadthFirst().

(1) The CPU class was called here to allow the BreadthFirst() function to work successfully when determining the path of the maze. The CPU class works identically to the CPU class in the MazeGenerator script.

(2) The AllCellDictionary() function works by constructing a “pseudo-grid” and adding each cell to a dictionary. In each iteration of the for loop, the cell script of the actual cell created by the MazeGenerator is found. The function here works similarly to the ConstructGridLayout() function in the MazeGenerator script. Some components (such as the instantiation of the actual cell in the game scene) have been removed, however, due to its nature as a pseudo-grid.

(3) The NodeCreation() function works by iteratively going through each cell I created in the pseudo-grid and using a collection of IF statements to determine whether that particular cell is a node (e.g. T-Junction cells must be nodes). The function also attaches the node to other node neighbours to create a tree graph structure using lists in C#. This attachment of lists is also done by using IF statements (e.g. if the cell has a left wall, then it cannot connect to the previous node). One of the issues with my current implementation of this function is that all vertical pathways are considered nodes, making this implementation less efficient than it otherwise could be.

(An aspect that I have to clarify is that I did not use the DFS algorithm for the CPU, in spite of writing about doing such in the Design portion of my project. As I was producing the code and researching data structures, I came across a forum that discussed the differences between the DFS and BFS algorithm in traversing a binary graph. One of the users stated that BFS would naturally find the shortest path of between the start node and the target node, thus I decided to implement the BFS algorithm instead. The BFS algorithm is essentially the same as the DFS algorithm, expect it uses a queue instead of a stack).

Ultimately, the creation of the CPU script was difficult as I had not come up with the appropriate solution to the script prior to development. This meant a lot of work had to be done mid development solving all the issues that I had not thought of. In hindsight I would have tried to create a working pseudocode first instead of eagerly heading to development, as it would have saved a lot of time that can have been used elsewhere.

**Review – CPU Script**

The CPU script has quite a few issues in spite of its success in working for my implementation.

One of the main issues with the CPU script is that the object is unable to backtrack. This inability to backtrack prevents the CPU from solving complicated mazes as the CPU is likely to stop at a dead-end.

Another inefficiency with the script is that the algorithm has functions that share similarities to one another (e.g. GetNoOfNeighbours() function is similar to the CheckVertex() function). This makes the script more inefficient as unnecessary lines of functions are being used for no reason.

Despite the code’s inefficiency, however, it is able to work for my implementation as I only need the CPU to traverse short mazes with little backtracking in them. In the final solution, I must try to successfully solve some of the inefficiencies of the script whilst also ensuring that the program works successfully.

**Final Solution for the CPU script**

**Development for the Character Movement Script**

**Purpose – Character Movement Script**

The purpose of the character movement script was to create a control scheme that allows the user to quickly navigate a maze. This movement script functions via the use of IF statement that determines which key is being held, and then the script moves the character object accordingly. The movement script functions similarly to the movement in Tomb of the Mask where the user cannot stop the character without colliding against a wall.

**Brief Development Description (e.g. how it went) – Character Movement Script**

The character movement script was created after finishing the maze generation script. The script had some initial issues during development as I had no prior knowledge as to how the rigid body component of the character object influenced other methods. I was, however, able to successfully create the script after some initial trial and error and learning about the rigid body component. Below is a prototype of the Character Movement script created via the C#.

**Prototype Description – Character Movement Script**

\*The character movement script functions through the use of the rigid body component native to the Unity engine. The Rigidbody.MovePosition() method allows the character object to move the position of the character in the desired location. I had initially used the Rigidbody.Position() method to move the object, but I realised that this method doesn’t render the intermediate positions necessary to move the object; the method just teleports the game object. I had also tried to use the Transform.Translate() function, but the rigidbody component of the object seemed to override this function, as the Transform.Translate() function did not take into account the physics of the rigidbody.

\*To implement the movement of the character, I had to use 2D vectors to move the object in four directions. This was done by using two vectors that described the horizontal and vertical movement of the character.

\*For the IF statements, I decided to use the Input.GetKey() method as it allowed the object to slide indefinitely in the maze via one input of the key. I had tried to use the Input.GetKeyDown() method, but I realised during testing that the movement of the character became more disjointed. I had to tap the key multiple times to reach a suitable distance in the maze, making it more of a hassle to go through the maze.

Later on during development, I decided to modify the Character Movement Script so that it could translate the character object to the starting cell. This was done by finding the starting cell’s vector position and assigning that value to the character object. The purpose of this modification was to make it easier to start the game when rearranging the game layout, as I wouldn’t have to translate the character manually at all during that process. It also ensures that the both Player One, Player Two, and the CPU objects are in the exact same point at their own respective mazes, preventing any potential bias.

Ultimately, the movement script was created to allow the player full control over the character objects. I based this movement script heavily on the attributes that were present in the “Tomb of the Mask” game that I researched in “Evaluation of Current Systems”. I had initially tried to implement a script that attached the character object to the rail grid like system to traverse the maze, but I then realised that a movement script like the one above functioned just the same in a maze setting.

**Review of the Character Movement Script**

**Final Solution for the Character Movement Script**

The final solution for the Character Movement Script includes the addition of comments and suitable variables to ensure that others can understand my code successfully. I also decided to adjust the moveSpeed variable a bit higher so that players had more speed. I realised during this adjustment that increasing the moveSpeed variable by too much would influence the T-junction mechanism of the maze negatively. The character object would move to fast for it responded on time. Due to this, I had to compromise at a moveSpeed of 1.3f.

**Testing**

| Test No. | Description | TEX (Typical, Erroneous, Extreme Input) | Expected Outcome | Actual Outcome | Corrective Actions |
| --- | --- | --- | --- | --- | --- |
| 1. | Selecting a menu option. (Does the selection with the menu option work?) | T: A menu option has been selected appropriately using the arrow navigation system. | The screen will transition to the appropriate scene. |  |  |
| E: No menu option has been selected. | No scene transition will occur. |
|  | Fading in and out transition. (Does the game transition between the scenes with the correct animation?) | N/A | The fading in and fading out animation will be called as appropriate whenever the user moves between scenes. |  |  |
| 2 | Modifying the settings. (Am I able to modify the settings of the game?) | T: A setting option has been modified appropriately. | The setting will be modified appropriately, and the setting changes will translate to the game scene. |  |  |
| E: No setting option has been modified by the user. | No settings will be modified by the user. |
| 3 | Moving the character object in PC. (Can I move the character quickly through the maze using WASD keys?) | T: The character object is moved using the appropriate directional character keys. | The character will move in the assigned direction and slide along the maze. The character object will stop when it collides with a cell wall. |  |  |
| E: No appropriate directional character keys are used to move. | The character object will not move. |
| 4 | Moving the character object in mobile. (Can I move the character quickly through the maze via swiping?) | T: The character object is moved using the appropriate swiping mechanics on the mobile phone. | The character will move in the assigned direction and slide along the maze. The character object will stop when it collides with a cell wall. |  |  |
| E: No appropriate swiping mechanics are used to move. | The character object will not move. |
| 5 | Moving both character objects simultaneously (multiplayer). (Do the movements interfere with one another?) | T: Both character objects are moved using the appropriate directional character keys. | Both character objects will move simultaneously along their own mazes. There should be no overlap between the control schemes. |  |  |
| E: No appropriate directional character keys are used. | Each players’ respective character object will not move. |
| 6 | Player collision with the end\_cell during the standard game scene. (Do I transition to a different maze?) | T: The character object collides with the end\_cell directly | The scene will transition to the next level after it completes the collision. |  |  |
| E: The character object attempts to collide with the end\_cell through one of the thin cell walls. | The scene will not transition as the character object did not collide with the end\_cell directly. |
| 7 | CPU collision with the end\_cell during the standard game scene. (Does the game end successfully?) | N/A | The scene will declare the CPU as the winner and transition back to the main menu scene. |  |  |
| 8 | Player collision with the end\_cell during the multiplayer game scene. (Does the game end after a player has collided with the end\_cell?) | T: One of the player’s character object collides with the end\_cell. | The scene will declare that player as the winner and transition back to the main menu. |  |  |
| E: One of the player’s character object attempts to collide with the end\_cell through one of thin cell walls. | The scene will not transition as the character object did not collide with the end\_cell directly. |
| X: Both player’s collide with the end\_cell simultaneously. | The scene will declare a random player as the winner and transition back to the main menu. |
| 9 |  |  |  |  |  |