A Level Computing
H446/03
Programming Project

Formby High School

Tower Defence Game Documentation

By Sam Spawton

Contents

Section 1

Section 2

Section 3

- **Core Modules with Testing**
- **Post Development Testing**
- **End User Testing**

Section 4

- **Success Criteria**
- **Stakeholder Review**
- **Strengths of Solution**
- **Limitations of Solution**
- **Maintenance**
- **Further Development and Improvements**

Section 1

Project Overview

The program I am intending to create is a simple tower defence game, it will be built around the player having to survive multiple waves of enemies by building towers and structures to slow them down and to defeat all the enemies. The program will use pathing algorithms to add difficulty and complexity which will be applied to the enemies in the game. They will use these algorithms to try to reach the player's main base, where if an enemy comes into contact with the base it will lose health points. The main objective of the game is to use strategy and micromanagement to prevent as many enemies reaching your base. If the player's base loses all Health Points (HP) then they will lose.

The game will be connected to a database of previous players and their corresponding scores; like a classic arcade game where each player has 3 characters to create a username and a score. At the end of each play through the score of the player will be stored in the database – replacing the same players previous score if the new score is higher and outputting back into the game the leaderboards of the highest scores which will be displayed in descending order starting with the highest score. The scores in the database will be analysed and sorted into order using another sorting algorithm.

I feel like there is a need for this software as there are many games that have similar mechanics but there are very few games with a retro arcade feel. The software will have the 8 bit graphics of a retro game such as Super Mario Bros for the NES and the complexity and difficulty of an intense real-time strategy game such as StarCraft or League of Legends.

As stated before there are not many games currently existing to my knowledge that are trying to combine these genres, I think my take on this project will create a surprisingly nostalgic game with a challenging undertone due the adaptive AI built into the enemies in the game. This will set it aside from many other games in its genre as it will be simplistic with its GUI and content but will still have the replay ability factor that all games must have.

The genre of tower defence in gaming is always adapting, from either using Base Building mechanics in games such as Dawn of War or Halo Wars to First Person Shooters combined with Tower Defences in games such as Orcs Must Die and even

Balloons. It is a very broad genre and has many different subgenres, which is why it's difficult to state whether something like this already exists; as the other tower defences may have similar features and mechanics but may be played with totally different objectives. So as difficult as it may be to state whether similar content exists or not, it is fair to say the ideas that I have for my project are not common in the current assortment of titles available but that means that there is no better reason to try something new to see if the combination works.

However there are some games with similar features, for example there are tower defence mobile applications such as Plants vs Zombies, console and pc games such as Orcs Must Die! And cross platform games such as Fieldrunners. These games are unique in their own ways taking different approaches to Tower Defence games. In terms of perspective being either; top down, first person or third person. They also have different approaches to levels by either having the game work with waves of enemies or various levels the player must complete to make progress.

I am going to try and create my software in a way that it is separated from other similar titles by taking inspiration from old retro arcade games, classic games from older game consoles and by researching new features that normally don't fit into this genre and seeing the ways I can adapt them to work in my design. By doing this I can create a totally different user experience to any previous tower defence games they have played before.

The target audience for the software I am creating is "traditional gamers", what I mean by this is a people from about the age of 16 upwards to around the age of 35. This age range typically contains the largest demographic of frequent gamers who play video games on a regular basis. This is my target audience as studies suggest the largest quantity of gamers are within that age range. I also found out whilst researching that there is a large Tower Defence game market ready to be capitalised on making this genre of game and this age range of gamers optimal for my project.

I will also aim to create this software to function on IOS and Android operating systems as such a large quantity of society within the 16 to 35 age range own smartphones. Making smartphone users an ideal target audience for my software.

My project was highly inspired by games I play myself, I have always enjoyed playing strategy games to challenge myself and enjoy the thrill of real time strategies where you must think quick on your feet to overcome the challenge the game provides. I wanted to implement this type of atmosphere into my game so the user felt a sense of urgency

and pressure to think smartly or they will be punished by the game. This came from a game I have played called Dawn of War Soulstorm. The game is considerably old being released in 2008; however the goal of the game was to create a base, gather resources to expand and upgrade while fighting either enemy players or enemy AI. The game is very challenging on harder difficulty and when you beat the game on those harder difficulties you get a sense of accomplishment that you finally beat it after hours of struggling, an aspect I would like to implement into my own software.

The ideas for my project are also heavily influenced by two other games I play. Firstly, the aspect of upgrading towers, levelling up from prolonged battle experience, intense scenarios where the player must use strategy and finally the punishment the game will reward you with for bad decisions comes from a game called FTL (Faster Than Light). FTL is a top-down space ship simulator, in the game the player controls the crew of a spacecraft which contains vital information to save the Federation. Throughout the game you must travel through sectors of space, fighting battles and upgrading your ship and crew to deliver the information you fought to protect from the Rebel Fleet. Combat takes place in pausable real time, if a combat is lost and your ship is destroyed the game ends, forcing the player to start all over again. This game heavily influences my project as FTL is a punishing game with intense difficulty, it is also a massive influence as one of its features is that throughout multiple runs you will unlock different spacecraft you can use in future runs.

The aspect of being rewarded for playing the game over and over is something I would love to implement into my game. This also comes quite heavily for the final game that influences my project which is called For the King. FTK is in early access but is not lacking in content or community interest. It is a challenging tabletop RPG where the player must complete a series of quests, upgrading their equipment and levelling up from completing quests, defeating dungeons or just winning the occasional skirmish with a cult of witches. The game is considered very difficult and is created so you can make each playthrough of the game more interesting and easier than the last by unlocking certain events, characters and items in their ‘Lore store’. In each playthrough the player will receive a certain amount of lore points based on their progress in that run, these lore points are redeemed in the lore store to unlock more in the next run that player has. Until eventually they can complete the game.

The stakeholders of my project will include my classmates, they will be involved in the testing and development of my software as they are all within the age range of my target audience and may provide valuable insight into the development process as they

are all experienced on the consumer side of the development process with video games and have experience in various genres and types of games.

Other stakeholders will be my teachers who I will need for their expertise in computing knowledge to help me identify the relevant system requirements that my software will have for consumers. They will also be valuable for the testing during the development process as they may have valuable insight into any issues I may encounter be it syntax errors or performance concerns. These stakeholders are referenced below with further information about contact info and their roles they will play.

I will also require other students within my year group to test my software and to provide feedback about balancing concerns (whether the game is too difficult or too easy), they will also be essential for identifying new requirements my software may need in order for it to be improved. The other students within my year group who I will be assessing my project with are also referenced below.

Stakeholders

The stakeholders of my software will mostly be my target audience; this will consist of the players of my game which will mostly be classmates, friends and other students. It will also include people who will help me test the features and structure of the game e.g. Teachers, frequent gamers and experienced people in the field of game design (QA contact from Lucid Games).

Other stakeholders may include companies who would like to publish my software, for example ‘In-Browser Games’ website owners who would like to upload my software onto their websites for their audience of players to use. This could be from websites such as Miniclip.com or Agame.com. This section of stakeholders may also include Developers from software such as Steam who oversee Steam Greenlight, where players can self-publish their software by pitching it to this developer’s team, if accepted the stakeholder would then become Steam Publishers who would oversee publishing my software.

Other stakeholders in my software may be other developers working on similar software; this could be other developers in my class creating a similar game. Or other developers creating software who are also applying for either Steam Greenlight or to be published on a Website of games as the better design would be the only piece of software accepted. There is a large pressure on developers to 1-UP their competitors as the Video Game market is constantly changing, new innovative ideas are constantly

being discovered to produce high quality, community loved games. Making the market for new software very competitive.

My software might also be interested in by forum editors or media sites who are wanting to post reviews of the game, starting community feedback for the software or wanting to raise awareness and ‘hype’ for its release and any following patches, updates or changes. Although there is slim chance of the software which I create making any revenue there is also the possibility for marketers to be stakeholders; people wanting to publicise the game and generate revenue for both themselves and the developers. If marketing was a concern then the distributors of the software would also then become stakeholders, concerned with age ratings, themes across the game that may be sensitive to certain players and anything that may offend certain groups of people. These distributors could be owners of video game shops or even online groups that would like to sell the game via virtual copies rather than physical.

The media may also have interest in the software as they may be wanting to post reviews of the software, giving it a rating from data collected by the community of players. If a review was to be posted of the game then they may inquire about any future updates for the game, any following games related to it and any opinions I might have as the developer of the game.

If the popularity of the final software is great enough that the game has interest from the previously mentioned stakeholders then the main people who will want to utilise the software will be those who can make revenue from it, this mainly consists of the publishers, marketers and new players seeking to gain a copy of the software. These groups will want to utilise the software by playing it and enjoying the content it has to offer. Marketers and publishers will also want to utilise the software by selling copies to groups of distributors, making revenue by sponsoring the game with certain brands and by getting more awareness for both themselves and the developer if the software is a successful.

The stakeholders of my software will benefit from the publicity generated by reviews, they will benefit from any revenue the game creates from either sales or sponsorships the game has within it. The software if successful should also benefit the marketers and publishers from the publicity gained, this would mean that more companies and developers would be aware of them and would seek them out for similar services as given to me to help generate awareness and a revenue from my software. The players of my software would also benefit them by giving them exciting new content to play and enjoy, providing stimulating challenges that would keep them entertained for hours.

Each playthrough of the game would offer a unique experience keeping the target audience as there would always be new features and challenges for them to overcome. Other stakeholders such as media, forum sites and game reviewers would also be supported by any success of the software as their name would be connected to a successful brand, raising awareness for them and providing revenue if they sponsor the software.

Each of the stakeholders would be involved in numerous ways with the software, for example; The players will be able to report any bugs or errors directly to me by using a built-in bug reporter they can access. This can then log any errors they report and send them to me the developer to resolve. The players will also be able to get involved in the software by giving feedback about their experience and any criticisms they may have about the content, allowing me to discuss with them likely future changes or tweaks I could make to make the software more enjoyable for them. These players will be able to report any issues with the software or request any changes whenever they like by posting them on a forum dedicated to the software where I can read through their thoughts and issues, depending on their insight into the software they have they may be able to get even more involved by having a discussion with me through my email address if I think they can add value to the software.

Other stakeholders such as the media and forums will be able to get involved with the software through the website that will support the newly created game. On this forums site they will be able to create a new thread to discuss with me any requests for my progress on the game and if I am regularly updating and planning changes, they will be able to read through the players threads and get their opinions on the software directly from the website and they will be able to remain in contact with me through the website by messaging me privately as I will connect my email account to the website so I am notified when important users have sent me a message.

Users of the website will also have to create accounts which will be stored on a secure database that I will have to create, this way I can give different accounts various levels of access on the web page. Allowing me to have moderators and administrators to ensure the website is used solely for its purpose without any offensive behaviour affecting the community. These moderators and administrators would also be stakeholders but mostly involved with the website rather than the software itself.
The marketers and publishers would be able to keep in contact with me directly through my personal email as their requests and issues may be deemed more essential to be addressed first before any of the community issues. This would not mean I would

dismiss requests from the community but the priority of the publishers and marketers may be more pressing.

Name: Sam Spawton
Occupation: A Level Student
Role as Stakeholder: Project Manager and Developer
Place of Work/ Study: Formby High School
Email: Samualjs@gmail.com

When it comes to stakeholders I will be in charge of development of my software and will be responsible for any changes that need to be made if the software quality is not as high as it should be. I will be the project manager and will compile reports of issues and changes that need to be made, as well as recognising when a significant milestone of progress has been made and functions exemplarily.

Name: Jamie Foggin
Occupation: A Level Student
Role as Stakeholder: Involved In testing and QA (Error and Bug reporting)
Place of Work/ Study: Formby High School
Email: Fogginjames1999@gmail.com

My main stakeholders for my software or co-students as we will all be developing software individually for a website we will be hosting, this is so we can all upload our completed computing projects onto a central website we will all contribute to and manage in different aspects. Ultimately the main stakeholders are ourselves for university applications to make us stand out against other students. Jamie's will be involved in the testing of my project and the builds that I will compile to help me identify major issues that cause stability issues in my final product.

Name: George Bond
Occupation: BTEC Student
Role as Stakeholder: Analysing software against success criteria
Place of Work/ Study: Formby High School
Email: George.Thomas.Bond@gmail.com

With my software I will have certain milestones, each with their own small set of success criteria. This will ensure I will not add more features to my software unless I have the available time and stop me from having a half functioning build. So each milestone I complete will have a fully working build that does everything it was identified

to do. George will be comparing the software against the success criteria in each milestone to see if it is up to the standard expected for the final product.

Name: Jude Taylor
Occupation: A Level Student
Role as Stakeholder: Peer review for GUI
Place of Work/ Study: Formby High School
Email: JudesHTC@gmail.com

As I am creating my software I must ensure that it is as user friendly as possible to make it stand out against other current systems that it will be competing against. Jude will help my identify any tweaks or potential changes I could make to the GUI to make it more user friendly.

Name: Andrew Riozzi
Occupation: Lead QA at Lucid Games
Role as Stakeholder: Identifying Performance Issues (Potential Lag Causing Issues)
Place of Work/ Study: Lucid Games
Email: andrew.riozzi@lucidgames.co.uk

As I work my way through the stages of development it would be invaluable insight to have an actual member of the game development industry to give me any advice on changes I could make or adjustments that would make the game less memory intensive on a system to lower the system requirements to run the game optimally.

Name: Robert Allan
Occupation: A Level Computing Teacher
Role as Stakeholder: Software Development Support (Analysing modules/functions)
Place of Work/ Study: Formby High School
Email: R.Allan@formbyhighschool.com

As a computing teacher who has significant programming knowledge when it comes to object oriented programming his insight into the structure of my software and how it is written will be necessary to ensure my code is easy to modify and change throughout the development process and to reduce any conflicts and issues.

Name: Daniel Rigby
Occupation: A Level Computing Teacher

Role as Stakeholder: Identifying potential performance issues
Place of Work/ Study: Formby High School
Email: D.Rigby@formbyhighschool.com

Knowledge about computational methods and how to successfully implement each method effectively into my program to guarantee efficiency and coherence across different operating systems Mr Rigby will be invaluable for the programming structures and algorithms I would like to implement into my software.

Current Systems

My design is a base building tower defence game where you will fight off waves of enemies before they reach your life points in order to win, each wave will progressively get harder therefore the player needs to plays towers and obstacles strategically in order to win.

Current systems similar to this involve other tower defence games that have similar features such as tower placing and waves. An example of this is the game series “Orcs Must Die” which I will be taking fantasy inspiration from and will be analysing to look at enemy pathing and turret placing. However my system will be different as the example above mainly focuses on a main character who goes round to place the towers and supports the towers where as mine will be solely based on the tower building and the redirecting of enemies in order to expose them to turret damage for longer. My system will not be “first person” it will be third person based with a view from above the battlefield where the player focuses more on the strategy aspect rather than the “Action/ Shooter” aspect.

In the example of the “Orcs Must Die” franchise it is commonly used by teenagers as it is rated and was marketed for that age range by its use of fantasy violence, blood and gore. These types of features typically appeal to teenagers. Lots of younger teenagers are the main target audience from the “Action” aspect to the game however the “Strategy” aspect attracts older teenagers and older people still as strategy games are enjoyed by many age ranges. Therefore the main target audience is likely teenagers, children and young adults however the fantasy aspects and other features may attract a larger audience of players.

The games in this franchise are typically accessed via games consoles and computers as the franchise has released 3 games across PC, Xbox 360 and PlayStation 4. The franchise has released 3 games; Orcs Must Die!, Orcs Must Die 2! And Orcs Must Die! Unchained!. The games have been extremely successful with Orcs Must Die rated 10/10 on Steam by it’s PC players, Orcs Must Die 2! Was also extremely successful with another 10/10 on Steam and 9/10 by IGN (In Game Name).

The game works by the player becoming a “War Mage” type hero who must place defences and obstacles down to slow down and stop waves of oncoming enemies using those defences and by engaging the enemies yourself as the hero. It uses a relative top down approach in proximity to the hero rather than the typical third person perspective to follow the hero around while the player moves the hero to potential weak zones. The orcs will path to “Rift Zones” which if they reach will deduct points from the player’s initial “Rift Score” whereas other types of orcs may deduct more points. If the player’s Rift Score drops to zero then they have lost the level and need to start over.

My new system will be better than this example by focusing much more so on the strategy of turret placing and the “Pathing Algorithm” the enemies will be instilled with to add more complexity and difficulty to each of the levels. This would be more suitable to the older ages within the target audience more so catering towards the “Strategy Games” players as the game will require quick responses to situations where the enemies quickly change their paths. The game will be more challenging as each wave progresses, stimulating the player to play more strategically and logically adding more depth to the strategy aspect. The base game will be designed to be relatively difficult in order to give a greater sense of achievement when a solution is determined, also if I am able to develop a random level design feature taking inspiration from other strategy games it will be more unpredictable making it yet even more difficult and more enticing to strategy players as they like to be mentally stimulated and challenged.

From the research I have done into the Orcs Must Die I have found many features I would like to implement and I have been Inspired with new and alternative ways to design and create my final product. I will try to implement the graphical 3D layout from the example given in a much simpler form as it is far too complex for me to develop within the given time frame but I will use the base Idea of Corridors, back pathways, obstacles and potential areas for optimal turret placement such as higher ground and bottleneck zones to add more versatility to the player's experience and more options for the player.

In the image above is gameplay from Orcs Must Die 2, one of the games in the OMD franchise. In the screen capture above you can see the player in the bottom middle of the game screen who will be running around the game map (as shown in the top right), placing traps for the enemies to get damaged by. The trap options the player can use are in the bottom middle of the GUI - each with their own cost of currency and their own ways of working. In the image above you can also see a translucent red preview of the “tiles” in the game field where the player can place traps. A feature I will hopefully implement into my own software but using a different 2 dimensional approach.

Other similar current systems are strategy based games and apps currently on the market for players, a good example of this which my game is influenced by is Plants vs Zombies. This mobile application became quickly popular with younger audiences using smartphones and tablets when it was released. The objective of the game was to fend off waves of different types of enemy zombies from encroaching on your back garden; so to defend it the player will place different types of plants to combat the zombies on a grid in the garden. As the zombies approached the backdoor into the house they would be attacked, slowed down by or be ambushed by multiple different types of plants.

The game had an exciting feel for it and quickly got audiences addicted to it, the game became very popular causing two more games to be created; firstly another mobile application being Plants Vs Zombies 2. This was an updated version of the game with more content but the same game mechanics, allowing the users of the initial software to have more content to enjoy. Secondly Popcap Studios (the developers of the application) teamed up with microsoft to create

a console build for the game which took more of a First Person Shooter approach where the teams would play as different varieties of plants and zombies in online multiplayer.

All of the products produced by Popcap were aimed at the younger audience. Reviews for the game gave the mobile apps an age rating of about 10+ and the console game an age rating of around 11+. The applications were used by young children at the time of release, with no doubt players both under the age rating and over the age rating as it was such a craze at the time. These products were accessible from most smartphones and tablets at the date of release and are still available on these platforms, the initial software was released on both IOS and Android making it readily available for the majority of people who wanted a copy of the games.

The software I am creating would be more suitable to the my target audience and stakeholders as I will be programming it with newer software that allows me to incorporate more advanced gameplay mechanics easier into my system with it causing less system strain on the device it is being used on as newer software is more efficient for memory and processing usage with particle effects and graphical animations. Meaning that unlike the preceding similar systems it would likely be smoother performance wise making it more enjoyable to players.

My new software would also be more suitable to my stakeholders as I can research into the reviews of the game and see the mechanics that people liked, didn't like and the mechanics people wanted to be in the software but were never implemented. By doing this I can make my software have a more user friendly GUI that is easier to use and understand making it more convenient for both young children who would be interested in the software and even the older generations who struggle to grasp the basics of new software and games. The reason that I am including older players into my target audience is because research has been done in recent years that show in certain countries that their are more and more players over the age of 50 taking part in gaming.

One of the other concerns with the Plants Vs Zombies franchise is the "scary violence" that it contains which has been pointed out in reviews such as:
https://www.commonsensemedia.org/app-reviews/plants-vs-zombies
From research into other reviews in similar current systems I will be able to adapt my game to have a better balance of fantasy and violence to ensure that the younger players aren't deterred from my software while still making it exciting and enjoyable for my older players.

I could use my research from user reviews of similar systems to adjust the play style and difficulty of my game, thus making it a challenging strategy game but with a versatile difficulty system embedded into the settings where the users can select how they would like to play the game. Providing lower difficulties for new players who are just learning the game and for the younger audience and a more challenging high level difficulty for hardened players who like a challenge.

In the image above you can see how a 2D approach to a tower defence game appears. This screen is a capture of one of the missions you will play in Plants Vs Zombies. On the left of the screen you can see all of the available plants you have to place in the garden to fight the zombies, across the top just to the right of plant selection screen you can see the currency used to purchase new plants. Right of the currency is a shovel to remove unwanted plants from tiles in the garden and in the GUI below you can see a hoard of zombies approaching from the right against the player's hand placed defence of plants on the left.

One of the other similar systems I have researched and analysed is Clash Of Clans. Clash of Clans is an online multiplayer fantasy themed tower/ base defence application, the goal of the game is you as the Chief of your village must build up your town and collect resources, these can be gained from attacking other players or from gathering them yourself from specialised structures (e.g. a gold mine). The players will form clans and communities where they can train and share troops, attack other players and clans or even just focus on the pseudo-single player campaign where they will siege a series of nearby goblin towns to gather resources.

The resources the player gains can be spent on new structures for the town, upgrading current structures or for training troops. This means that the player will initially start with nothing and must pillage and fight their way to promotion, players will also have to build their village strategically as it can come under siege at any point from real live players across the world. So they must carefully place defences around their resource storage units to prevent them from losing their resources.

This image above shows another example of one of the current systems I am analysing, this shows another one of the maps available for the user to play on from the Plants VS Zombies game. This image shows that the gameplay can vary with different types of towers and enemies as well as different terrain. I would potentially like to add the ability to have multiple maps and a large variety of towers the user can choose from.

From a poll on the developer's website about the age range of players there is varied results. Showing that the community is dominated by teenage players while not far behind on the statistics is players up to their twenties and even players in the age range of 50 - 99.

View Poll Results: How Old are you (If you play clash of clans only)

Voters: 169. You may not vote on this poll

1-10 years old	2	1.18%
11-16 years old	69	40.83%
17-24 years old	37	21.89%
25-34 years old	23	13.61%
35-49 years old	29	17.16%
50-99 years old	9	5.33%

Source:
https://forum.supercell.com/showthread.php/277511-How-Old-Are-Most-Clash-Of-Clan-Players

Clash of Clans is a mobile application and was only released on mobile platforms such as Android and IOS. Therefore most players access the game from their smartphones, tablets or kindles. The game is free to download from the respective mobile platform applications' stores and can be downloaded by anyone. Meaning the game is accessible to whomever would like to play.

My software would be more suitable to my target audience rather than this current system as my software is going to be utilising the real time strategy aspect much more than Clash of Clans as in that current system Real Time strategy isn't very versatile even when sieging an enemies base. (The only point in the game where real time strategy is used). The issue with clash of clans when it comes to real time strategy is it is largely dominated by the pathing algorithms the game uses for where the units will move and what they will target, making it difficult to be precise and strategic with your attacks. My software will have a much more user friendly experience when it comes to real time strategy, they will be able to precisely place traps and blockades on the battlefield using a more user friendly GUI.

However I would like to implement the feature of upgrading structures and defences into the game using currency generated from killing enemies and completing missions. Another feature I would like to implement from Clash Of Clans is their questing system. In the application if you reach certain milestones or complete certain challenges the player is rewarded with resources and the idea of that mechanic would function quite effectively in my software. However this is a low priority game feature I may not be able to add into my software as effectively as I would prefer.

In the image above you can see an example of a village someone has created. In their village they have their resources storage within the walls as well as defences behind these walls, along with traps outside the walls. As you can see there is a number of different defences and structures the players can place in the game grid allowing for a different experience every time.

This screenshot demonstrates the ingame experience for one of my current systems that I am analysing and shows an example base from a user when they have been playing Clash of Clans. From the screenshot above I have looked at the style of GUI that I plan to create for my game, this shows resources and money across the top of the screen as well as key buttons and a smooth menu interface across the game.

Current System	Advantages	Disadvantages
Orcs Must Die	Combination between Tower Defence and Shooter – Good for people who like strategy and those who like to tear apart enemies on their own Humour in the content that made it enjoyable by players – witty commentary in tutorials Vibrant graphics and higher quality animation Possible to purchase premium quality content without microtransactions	Character class choice can be compromised if you do not select character of choice before someone else does Balance issues with certain enemies not taking enough damage No need for strategy with the placement of traps, no need for mazes (major mechanic) and takes away from the tower defence feel Game feels to grindy and takes away from the enjoyment

Plants Vs Zombies	Compromise in the stereotype of violent video games, takes a non-violent, kid friendly approach Lots of content; 5 groups of 10 levels each with their own characteristics Different game modes such as Adventure, Minigames, Puzzle and Survival	Exposed violence to a very young target audience and desensitises them to violence Zombies are a scary aspect of the game and can upset younger players as they are a large portion of the players Each wave while slightly different has a similar feel with a grindy start and anticlimactic end
Clash Of Clans	Game becomes addicting and users felt drawn to keep playing, upgrading and rearranging their base High quality graphics and animation for a mobile game making it exciting to play Endless possibilities for base design making it very strategically driven	Game has been made unnecessarily hard to encourage players to spend money via microtransactions Game relies on waiting heavily for upgrades, resources and training. Meaning the player cannot sit down and keep playing they must wait hours or even days before progress is made

Computational Methods

Backtracking in computing has more than one meaning, In some types of software backtracking can mean trying out a sequence of actions and determining how far it will succeed until it becomes apparent that you cannot succeed. Or backtracking could be

used in the sense of level design where the player cannot get to a certain area or stage in the game at a certain time until a set of criteria is satisfied, then they will be granted access.

In my software aspects of both of these definitions will be used. For example in some of the more difficult waves in my game the user might have to face different types of enemies that cannot be killed by “conventional means”. By this I mean I will have enemies that can only be damaged by fire turrets, and the fire turrets may only be unlocked with either a large quantity of ingame currency or the more likely way I will implement it (By using For The King game mechanics) is by making the fire turret only unlockable to the player when they reach wave 3, meaning the user must play multiple games and succeed in each of them in order to proceed. To expand; in the user’s first game they may reach round 3 but will not be able to kill the Boss without a fire turret. Then at the main menu Turret Store the fire turret will only be unlocked after the user reaches Wave 3, 3 times. Therefore utilising backtracking the structure of the game for the player to complete the criteria of reaching wave 3 3 times before being able to progress any further.

Data mining is where large sets of data will be analysed by a computer system and over time patterns will emerge, showing relationships between certain fields of data. I will incorporate this into my software in a relatively simplistic manner, throughout each wave the software will analyse the types of towers most commonly placed by the player. As each tower will be effective against different types of units the game will then try to outsmart the player. It will do this by changing the structure of the next enemy wave, making the game send enemies that are affected less by the commonly placed player towers. This will also be one of the main sources of the system adapting itself against the patterns it analyses from the player if the player becomes predictable.

Computational methods are the various types of mathematical approaches to solving a problem and completing a task in computing. By using these methods, a system can analyse collected data and work backwards to find solutions, use best fit algorithms to find optimum solutions, use past successes to guarantee a working but not optimum solution, it can model scenarios by using previous outcomes by creating simulations and many others.

Simply put, these methods can be used by a system to recognise problems and constraints by analysing the criteria for success and by using the resources it has access to in numerous ways. After realising there is a problem the system can then decompose the problem into smaller modules, these can then be tested individually by

the system to pinpoint the issues by using divide and conquer algorithms or by using abstraction to remove all unnecessary aspects of the software, showing a simpler picture where it is easier to observe where any issues may be occurring.

Thinking Abstractly will be used in my software by the enemies in my game, they will be using pathing algorithms to get to their objective – the player's base. This means that they won't view the game field as the map that the player will see, the enemies will only be able to recognise a significantly simpler battlefield made entirely of a grid of squares. The enemy AI will then use shortest path algorithms on the grid to path towards their objective. Abstraction is used to make it easier for these enemies to move easier.

My software will also use the method of Thinking Ahead by determining the required amount of attacks one of the towers will need to make to destroy an enemy. By knowing how much damage an enemy can take before being destroyed and how much damage a tower can deal per attack the system can calculate the amount of attacks a tower must make, this makes it less intensive on the system as the number of required hits is already known and is stored in the cache for easy access.

By Thinking Procedurally, the towers in the game will identify if an enemy is in range, if an enemy is within range then it will fire a projectile, the tower will then check if the enemy is still within range to shoot, if it is then it will fire another projectile. This will repeat until either the enemy is destroyed or is out of range. Then it will repeat the previous steps to try and get rid of enemies within the vicinity of the tower.

The software will utilise Thinking Logically when a tower must decide which enemy to fire a projectile towards. This will occur when multiple enemies are within the radius of the towers range. In this scenario, the tower will use an algorithm to identify the most threatening enemy, it will do this my obtaining the distance from an enemy to the player's main base and compare all enemy's respective distances, the tower will prioritise the enemy which is closest the player's base which is also still within the range of the tower.

The last computational thinking method my software will be using is Thinking concurrently. It uses this aspect in the system by having multiple towers having to destroy multiple enemies all simultaneously. This means that my system will be prioritising the most threatening enemies to the player's chance of winning by having each tower update every frame, this way each tower is individually targeting enemies and destroying them.

Key Features of a System

My system is going to be able to allow the user to allow the user to play through multiple rounds of a tower defence game. There will be a main menu screen where the user can select the options to enable or disable the music, they can select the tutorial option on the main menu which will provide a brief overview of gameplay mechanics to teach them the basics or they could select Single Player where they will attempt to survive multiple waves of enemies. The single player option will be the main content of my software.

In the ‘Single Player’ mode, the player will be able to place towers in certain positions on the battlefield to slow the invading enemies and destroy them. Once an enemy has been destroyed they will be rewarded with gold to spend on other types of towers. After a wave has been completed another more difficult wave will start. This repeats until the base of the user is destroyed be it after 2 waves or 20, the game will adapt itself making it harder and harder. After this point the system will prompt the user to input their score into the database of scores where it will be sorted into ascending order.

The system will allow the user to select a variety of towers from a user friendly GUI which will then highlight positions on the map for the user to place the towers. Each tower will have the option to be upgraded once clicked on but this will most likely be used for strategy when all optimal tiles to place towers have been filled. Allowing the player to still progress even when they have limited tiles left.

There will also potentially be abilities that the user may use which will have large cool downs such as Earthquake where all the players towers will become less effective for a small period of time in exchange for damaging all enemies currently on the battlefield. Abilities like this are currently not the highest priority and would just be bonus features if their is left over time in the development process.

The player will also be able to access a Tower Store at the main menu between different rounds (not waves) where they will be able to unlock new towers that they will be able to use in future playthroughs, by adding this feature the game has more replayability to it and can be enjoyed over multiple runs with it adding more content the more you play. The players will not be able to identify what the new towers will be until they are unlocked adding a sense of mystery to the game which should hopefully encourage players to keep playing in order to satisfy their curiosity.

The system will work by the user selecting the Single Player option at the title screen to start the game, there will be functions in place that allow the user to select this using either the mouse, arrow keys on the keyboard, wasd keys on the keyboard and potentially via touchscreen if I have extra time at the end. During the game play the main user input will be done by the mouse using left and right click however I may add zooming into the game if I have excess time.

By allowing multiple inputs the system will work more efficiently and helping it cater to multiple types of players or even people with limited mobility functionality. In the actual tower defence section where the user is playing the game they will be able to navigate the tower selection menu at the bottom of the screen using the arrow keys or wasd as well as the mouse to add versatility to the players so they can enjoy the game no matter their preferred play style.

The key features my system will have that will make it stand out against the other current systems will be based around a much more user friendly GUI, my software will allow users to navigate the game environment more smoothly without rendering issues as it will take more a retro graphics theme. The retro graphics theme will also make it stand out as their are very few 16-bit tower defence strategy games out there, and as time goes on the old arcade games and N64 like games are becoming more and more popular as people are starting to enjoy the classics. Meaning that my game will appeal to users looking for an arcade like nostalgic experience.

Peer Research

I asked some fellow students in my computing class and around my college some questions about my project to get insight into mechanics they would like to see in a game similar to the one I am creating.

Do you frequently play strategy games?
- Yes I think that they are underappreciated and the game market should be more focussed on strategy based games that stimulate a player. (Nathan Witty)

- Mostly turn based strategy games such as Hearts of Iron or Sid Meier's Civilization games.(Ben Hood)

- I play nothing but strategy games, nothing is more enjoyable than defeating your opponent intellectually rather than a contest of reaction times like in FPS's (Joseph Ventre)

What key features do you think a strategy / Tower defence game should have?

- Some kind of strategy and variability layer, such as varied wave composition, enemy types, tower types and enemy types (Nathan Witty)

- I think that Boss waves should be incorporated to challenge the player at decreasing intervals to step up the difficult. Also enemies that are capable of circumventing specific defences (Joseph Venter)

From the insight provided from my peers I have adapted my Key features in order to expand my player base to make it more enjoyable for a larger spread of players. This will help me ensure my project is worthwhile and that it will have mechanics that are enjoyed by players who are more experienced in these games than I am.

Key Features:	**How it will be used:**
Tower Store	The tower store will be embedded into the system by being at the home menu of the game when it is first launched and when you are returned to when you have finished your attempt at single player. For each wave, the user completes they will gain 1 tower token that can be used in the tower store between runs. New towers and defences that the user can unlock in the tower store will cost different amounts of tower tokens, meaning the user must play over and over to unlock them all. Once a has unlocked a tower in the tower store it will become available in the tower selection bar at the bottom of the game screen in single player for the user to select and place.

Settings Menu	The settings menu will be able to be viewed from the main screen. The user will be able to select the menu by pressing “Settings”. This will open a new window that will allow the user to adjust the volume of the game music that will be playing in the background, or turn it off completely. They will also be able to adjust the game volume (tower shooting sounds and enemy death sounds). In this window, there will also be the option to change the difficulty from Easy to Hard. These two settings will affect the gameplay quite heavily so there is a significant step up in challenge to the user.
Tutorial Screen	The tutorial screen will be able to be selected from the main title screen by the user, this will open another window that will show an example single player scenario and will annotate the GUI on the screen explaining what it is and how the user can use it briefly. This won’t give the user a complete walkthrough but will point them in the right direction to learn the basics.

Tower Selector	The tower selector will be in the bottom of the GUI in the actual game when the user has selected single player. It will have arrow keys at the side of the selector for the player to view other available towers and defences. It will also be able to be navigated by the user using either the arrow keys or the wasd keys as they will not be used by anything else in the single player screen.
Scoring Database and Leaderboard	The scoring database will be accessed remotely through the actual tower defence software. When the user is defeated and has got as far as they can a screen will come up and prompt them to input 3 characters that represent their name or player ID, this will then be written to a text file within the same file location. Then when the user is back on the title screen they will be able to review the scoreboards for all players who have attempted the game. The software will interpret the text file and will read the necessary data and will compile a simple leaderboard GUI.
Game DataMining for Player Patterns	This feature will be embedded into the enemies spawning function. It will run quietly in the background checking if multiple towers of the same type are being placed. And if a pattern is detected it will modify the variables for the types of enemies that will be spawned.

User Requirements

In order for the system to be successful the user must be able to select all of the options of the main menu screen and must be able to successfully go onto single player and play the game. They must be able to place towers in all of the available grids on the battlefield, the towers must be able to target and attack the enemies that are travelling through the path set for them. The enemies must also spawn correctly and only move on the path set for them. For the game to function properly the waves must successfully tick over and get progressively harder as they complete more and more waves.

Considering the stakeholders are me and my peers for our university game website project my software must be able to stand up to competing software, it must stand out in its own genre and attract an audience. By my software being of better quality it would mean that it is more likely to attract more players to our website, and as the goal is to get larger website traffic and to grow into a small game website where people can upload their own creations for everyone to play. My software will be successful as long as it functions properly and is addictive for players to keep playing.

The hardware requirements for my software are not too intensive as the game will be running on relatively low graphics with limited animations in order to maximise the quality of the game content. Because of this the hardware requirements are relatively low. The software will be run in the Love2D game engine written in LUA, it would be fair to assume because of the content of the system would run comfortably on 1000 MB Ram minimum, a 2.5GHz dual core processor, 500MB of system memory on the Hard Disk. A keyboard and mouse would also be considered essential for this system as well as a monitor.

Software requirements for this software would include the LOVE game engine installed on the system if the final product is a .love-file, however it can be exported as windows executable file which will create a fused game. It will also require any of the following; windows 7, windows 8, windows 10 or Mac OS X.

Input requirements would be a normal qwerty keyboard which must include either the arrow keys and or the wasd keys, the user will also require a mouse in order to input into the system as I will need both of these in the development process to test functionality and to edit and modify the code.

My design requirements are a criteria of functions that the stakeholders would like the software to have which have all been mentioned above in Key Features. However it would also be preferable to have a brief GUI design for me to create the system around so it appears as the stakeholders would like.

The processing requirements for this are not intensive and will only require a simple CPU with a clock speed over 2 GHz, multiple cores not necessary but in all software cases are preferred.

The system will also not require much RAM for processing as it will only be storing a few sprites in the memory as the game is running so around 500MB or Ram would suffice however it would be preferable to have around 1-2GBs of Ram to be safe.

The output requirements of my software are most definitely a graphical interface of some kind, most likely a computer monitor, tv screen, projector or anything that the computer can send video to. The game will also have audio so while speakers, headphones or other means of transmitting audio would be better for the game experience they are not required to run the game.

The stakeholders being myself and my peers have all agreed on these requirements as a minimum and think they are all appropriate. Also I have discussed these requirements with other peers who are not directly involved in the development process and they find them agreeable.

Success Criteria

To evaluate the success of my software at the end of the development my peers and I will each fill out the success criteria stating whether each of my individual goals was achieved and how far that success goes. Throughout the development I will refer back to the success criteria to identify areas that were successful but not fully operational, by doing this I can make progress on the success criteria itself as well as repair any modules to bring them up to a higher standard. My peers will also be writing small reviews on certain modules that they have been designated to allowing me to evaluate all of the functions in my code individually; knowing which ones work 100% of the time and the ones that do not. This will allow me to solve these problems more efficiently.

Primary Goal	Level Of Success	How it will be evaluated
Allowing the user to access the settings menu from the home screen and change settings.	**High** - The user can modify all settings in the menu precisely.	This will be tested by seeing if the users change to the settings stay changed after these 3 intervals; they return to the menu, during the game and after the game. This will be tested in a quiet room with headphones to see if there are any noticeable changes to the
	Med - The user can change most of the settings in the menu.	

	Low - The user can change a few of the settings in the menu.	volume after adjusting the settings. It will also be tested by multiple different players to see if there is a change in difficulty when switching between easy and hard.
Allowing the user to redeem tower tokens in the tower store to unlock new towers	**High –** The user will receive tokens after each playthrough of the game they reach, the user will be notified when a new tower can be unlocked, the user can spend tokens to unlock a new tower, the tower will be active in their tower bar now and they can use it in battle.	This will be evaluated by multiple players reporting back how the tower unlocking mechanics are working involving the tower store to see if it is all functioning as programmed. Players will also be “stress testing” the game by using only new towers and lots of them to see if it causes performance issues or errors. Players will also test if all towers can be unlocked and that they all can be placed and work respective to their traits in the battlefield.
	Med – The user will receive tokens after a playthrough, at least 3 new towers can be unlocked before the game encounters issues, the tower can be placed and causes few issues in the game.	

	Low – The user will be awarded tokens appropriately and can unlock at least 1 new tower that will be usable but not perfect in game.	
The GUI must be easy to use and can be briefly explained by the tutorial button on the main menu	**High –** The system will allow the user to access the tutorial button to teach them the basics of the GUI. Also, all features such as buttons and menus must function as the user expects.	This will be evaluated by players testing all the buttons on both the main menu screen and within the game itself. Ensuring that the user can use them accordingly. The users will also have to view the GUI help screen which the game will offer on the main menu. The users will also need to test if simpler forms of this system work whether they are text based tutorial screens or not. The players must also be able to navigate the GUI effectively reporting any issues with functionality that they may encounter.
	Med – The system will allow them to see a tutorial screen at some point in the game that provides knowledge about the GUI. The system must also allow the user to use most of the buttons and menus on the GUI while are using the software	
	Low – The system will at least provide text based instructions at some point to describe the GUI to the user. They must also be able to access all essential buttons and menus on the GUI	

The program will provide a challenging experience as the player progresses through the waves without any overwhelming balancing issues with difficulty.	**High –** The system will allow the user to play through the game without it crashing or breaking, they will be able to enjoy the game and not be beaten my unfair game mechanics originating in the code.	These mechanics of the game will be analysed by having players try to beat the game over and over on different difficulties using destructive and abnormal testing to see if they can find flaws in the system which could cause it to break. The players must also ensure that the difficulty curve as the player progresses is not to sudden and does not become too steep and by doing so making the game impassable from that point. Performance will also be monitored by the players and they will report any graphical issues and issues with mechanics that could potentially cause a game crash.
	Med – The system must allow the player a relatively fair experience whilst playing the game, the game will not encounter any key issues that directly affect gameplay and will be able to succeed even against small balancing issues	
	Low – The user will be able to play the game for at least a few waves before being outmatched by the adaptive AI, they must also not encounter any issues that completely stop the game from running.	
The user will be able to have their score stored in the scoreboard database and will also be able to view other scores and the top 3 scores in the database	**High –** The system will allow the user to enter a 3-digit name to accompany their score in the database, they will be able to access the database through the game and be presented with the top 3 scores	These features will be evaluated by myself and my peers, we will each play the game to a different wave number to test if the scoring is working appropriately. We will also analyse if the scores are being stored in the database correctly and

	Med – The user will be able to submit their score into the database with a random string generated by the system to represent their player ID so they know it is their score. The player must also be able to view the database of scores through the software and see scores in descending order.	that they are being sorted correctly. Then finally we will observe whether the software the user is using will successfully identify the top 3 names in the scoreboard and if the system will output them through the GUI to the user
	Low – The user must be able to upload a score into the database and will be able to view the database by accessing it from the file location it is stored as it is unable to be accessed by the software.	

The success criteria described above has been decided from the discussions with my stakeholders Robert Allan and Daniel Rigby, the success criteria has been agreed to concisely cover an achievable amount of functionality for my final product. We have also covered the levels of success for each criteria and have decided on certain points that must be covered by each module.

Limitations

Throughout my project I will face many limiting factors that will slow down progress and reduce the overall quality of the final product. For example a major limiting factor will be available time to complete all aspects of the project that I have addressed above. I will struggle to manage my time as I will need to be simultaneously continuing with the documentation for my software, developing the software itself and continuing with my other subjects. This means I will need to allocate my time carefully in order to ensure the development of my project stays within a reasonable time frame so I can complete all of the game content mentioned above.

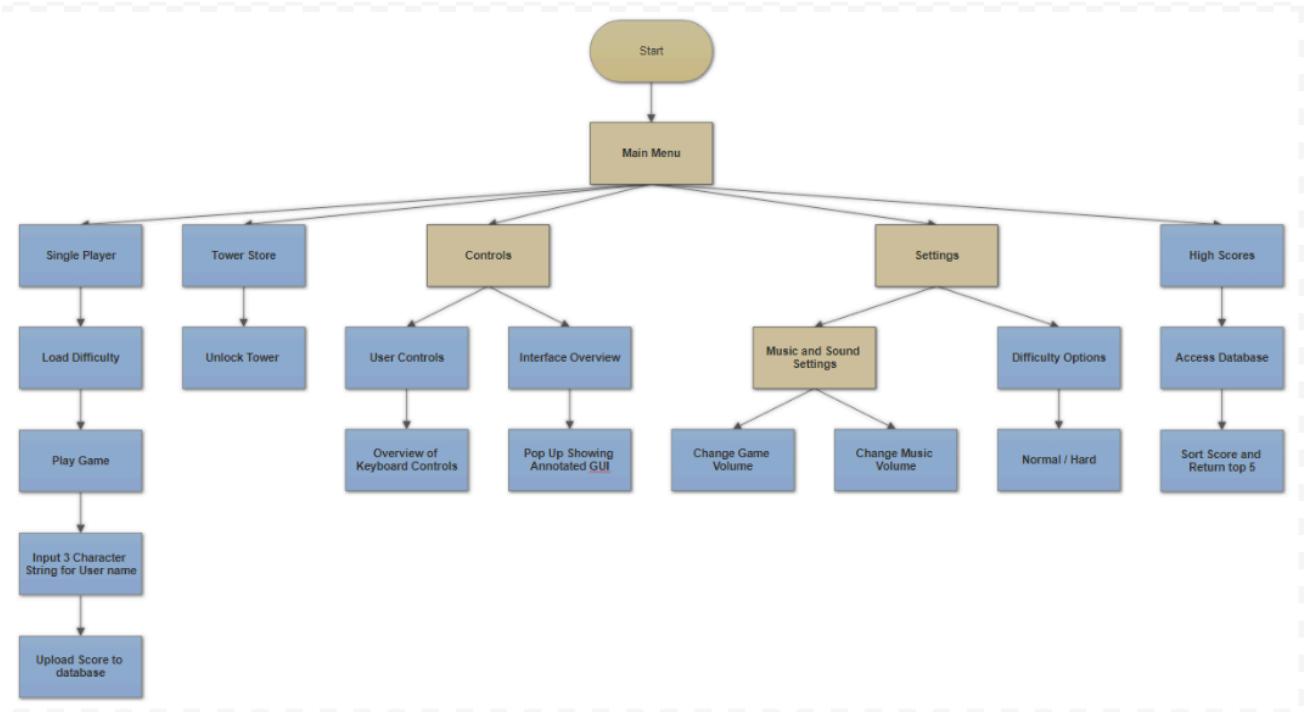
If my software is not fully functioning within the time frame allowed my stakeholders will be let down as I cannot contribute to the group project on time, delaying the progress of the project which would be a disappointment to my peers.

Another limitation is my programming knowledge of unity, this is a completely new programming language to me as I have only ever used Python and a little C#. Because of this I will take a few weeks learning the skills necessary to complete my project effectively with a more efficient program.

It is also possible that throughout the development process the requirements might be changed and adapted to suit the time constraints and programming knowledge issues that I may encounter, because of this I will have to modify my documentation and re-evaluate my software as a whole. This would be another limitation because my project will come to a standstill at these points.

Section 2

Top Down Diagrams



The diagram above shows an overall format that I hope to create with my programming project, the diagram shows the features I would like to implement into my software and how they will be nested within the main structure of the game. I would like to have multiple different layers of depth within each feature which are shown by the depth of the tree diagram.

The diagram is a rough first design for what I will hope to create as the basis for my program and my final product may differ from this slightly in some areas and more so in others depending on the issues I encounter while creating it.

To briefly discuss the diagram, I will start with the Single Player aspect; my software will be a game that is played by 1 player a time and the main content of the game will be loaded when the Single Player functions are called within the respective game and level managers that I will use to structure my code. Within these functions will lie the main complexity of my software, In here I plan to implement the A* pathing algorithm which we learned in class to allow enemies in my game to move across the game world and path around obstacles. The A* algorithm will be programmed to be applied to all enemy sprites that are spawned to get them from the start point to the end. The main concept of the game is for the player to place towers to prevent the

enemies from reaching the end before the enemies destroy the base by taking away health points.

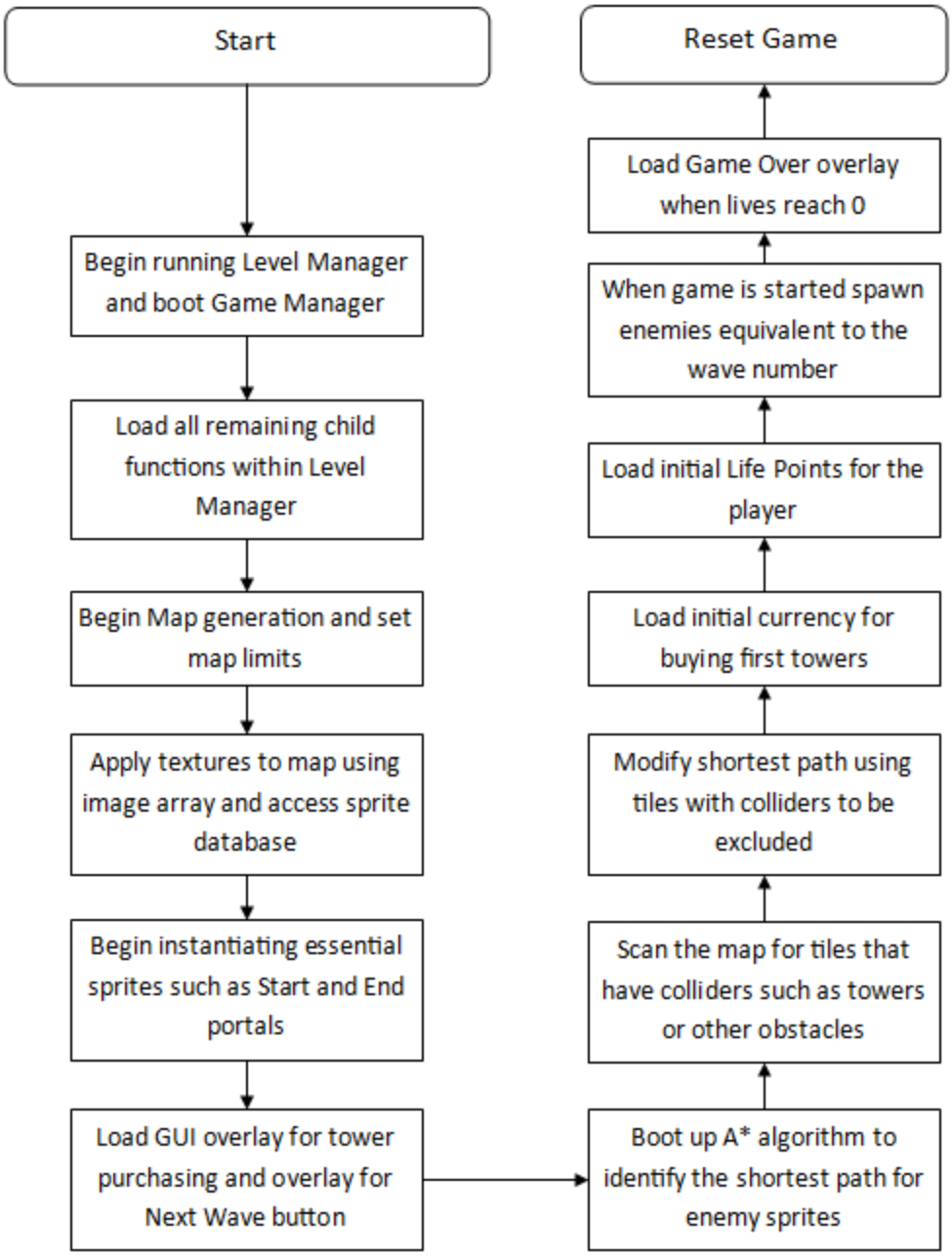
The next feature I will add is a tower store that will either be a menu that can be accessed during the game or before the game starts. This will allow the player to unlock towers and place them in the game world. Depending on programming ability will decide how I implement this but it will be an essential part of the complexity of my game as it will allow the user to access an array of different towers that will be stored externally.

Some more features I would like to add are menus within the game that allow the user to change the keys that are already set for things such as camera movement and other controls. This would allow them to rebind these keys before they start the game for a better user experience or to help people with limited motor function to play the game more easily. Within these functions I would also like to implement a way to show the user graphically which keys are used by the game with a brief explanation of their uses. This isn't necessarily a priority but it is a feature I have included in case I have more time than expected to complete the program.

In the game I would like to have some settings that can be changed to change the game experience, not in terms of usability but more so in terms of content. These settings would likely include a way to change the game volume, potentially in game music volume and if I have the time to implement the feature; difficulty. This would allow for more complexity and could add a more challenging experience to play the game.

Finally if possible I would like to add a way to store the players scores after completing the game. This would hopefully be accessible from a menu within the game so the user could see the top 5 highest scores for example. This is a feature I would like to add as it shows the program can store data externally which it can access again, this would also allow opportunity for usage of a searching algorithm such as a binary search to sort the scores from highest to lowest. This would make it easier to display the highest scores as it could just read the top 5 from the list rather than searching through each time. New scores could also be added into this list using an insertion sort.

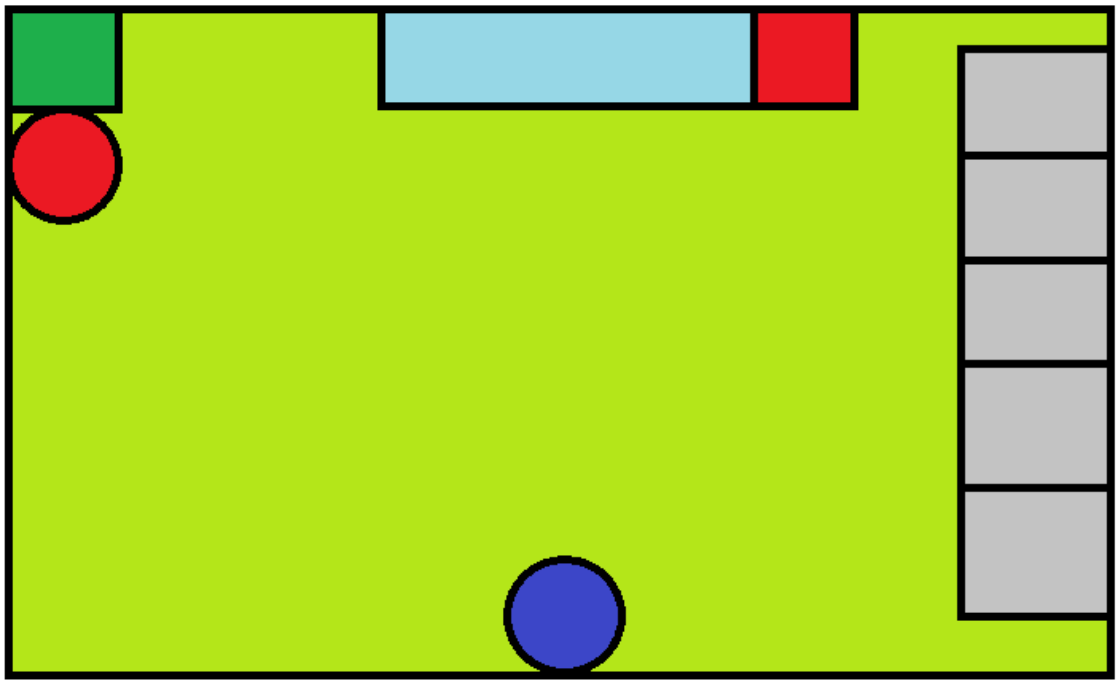
Data Flow Diagram



The above diagram shows the flow order of my program, this describes the boot order of the main functions and procedures of my game. This is roughly how I would like the program to run. Using this format I can plan ahead for future procedures while programming and make sure variables are private or local respectively when required.

Having diagrams like this with my top down diagram give me a clear idea how I will structure my programming and my functions and allows me to access and modify my code more easily as it will all be organised. This is useful for maintainability and error handling.

GUI Design



The above image is my first simple design for my GUI, this is how I would like my overlay to look for my game, In this section I will discuss the parts of the UI and what they represent within the game and this will allow me to have a graphical structure to follow when creating my overlay for my game and will hopefully give me an idea how the scaling I will need to use for the overlay.

Firstly I will discuss the red and blue circles, these are not part of the UI but represent 2 fixed points within the game world, the two points are the starting and ending points respectively. Enemy monsters will spawn out of the red portal and will path towards the blue portal around towers and potential hazards that may be implemented in the later stages of development. These portals will be able to be moved using a map file that is loaded to direct the map generation.

Next there is the green and red square. These will be a graphical representation of the players in game currency and their lives. These are two vital variables that are essential for the game, the green square will show the currency and will be a graphical representation of the currency variable. This will allow me to display the users money and will be updated in real time when the user purchases and places a tower.

Then there is the blue box in the centre top of the GUI. This represents the next wave button that will allow the user to start the next wave. When pressed it will execute the functions bound to the button and will begin enemy spawning and will prevent more towers from being placed as that will only be allowed between waves.

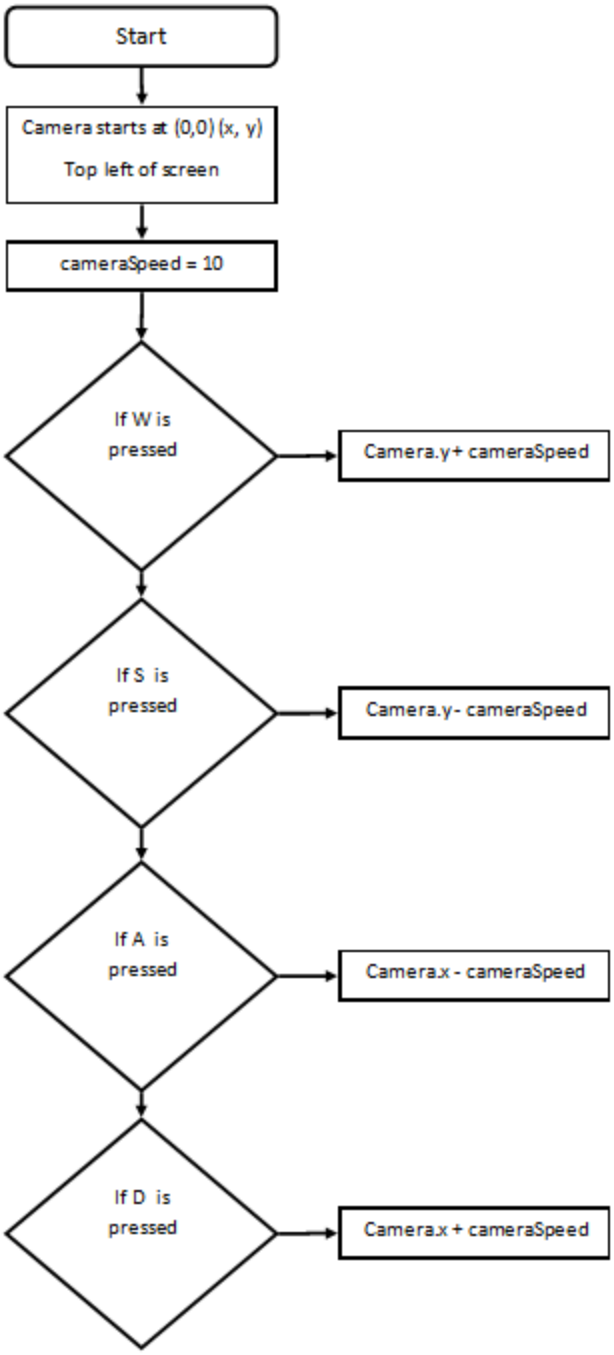
Pseudocode

Camera Controls

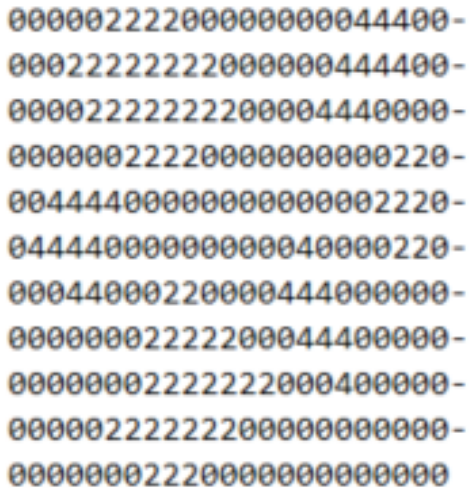
Camera movement (applied to main camera):

```
cameraSpeed = 10;

Private Function Input()
    If input key == W
        Translate (up by cameraSpeed)
    If input key == A
        Translate (left by cameraSpeed)
    If input key == S
        Translate (down by cameraSpeed)
    If input key == D
        Translate (right by cameraSpeed)
```

This module will be responsible for allowing the user to navigate across the game world by moving the camera position using the keys: W,A,S,D as if they were arrow keys, these keys are used as it is the standard for modern day video games.


Start
Camera starts at (0,0) (x, y)
Top left of screen
cameraSpeed = 10
If W is pressed
Camera.y+ cameraSpeed
If S is pressed
Camera.y- cameraSpeed
If A is pressed
Camera.x - cameraSpeed
If D is pressed
Camera.x + cameraSpeed

Level Grid



This image is a grid that shows the tiles that will be in the game, each number represents a type of tile which it will be allocated from an array of tile images. The map will generate using this level file and can be easily modified. This file is stored externally and is accessed by the program, this allows for future maps to be added and modified and allows for a multi-level game where there are multiple maps. The potential for multiple maps in the game is determined by time limitations and the game will still function with only one level.

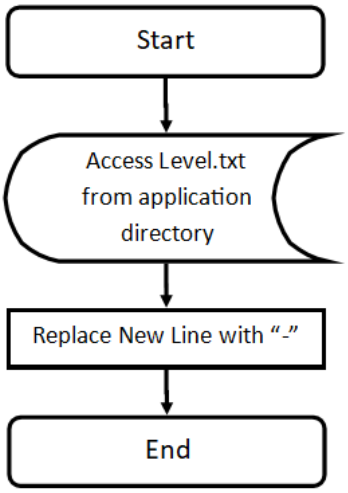
Private Function (Read Level)

data = Open level.txt

data.replace (newline with “-”)

data.split(“-”)

Separates Level file into rows and columns



Camera Limits

Camera limits (using map limits):

private float xMax

private float yMax

Public Function SetLimits()

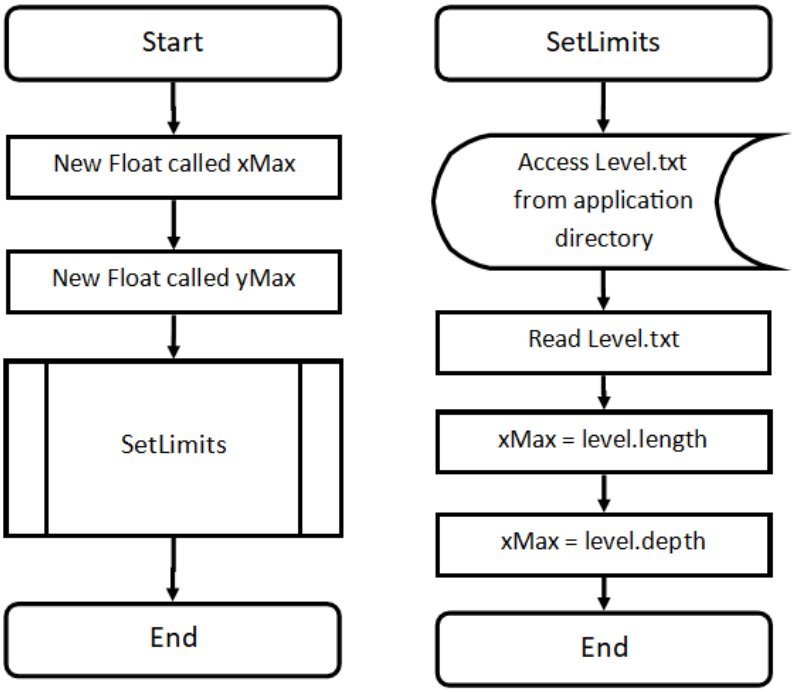
Read level file

xMax = level.length

yMax = level.depth

Close level file

This module will ensure the camera cannot be moved further than the limits of the game world, this prevents the camera from looking at the blank background behind the game world and prevents graphical issues and issues where the camera moves too far from the game world and they have "lost" it.



Level Management (map generation)

Private Function (Create world):

Tiles = new dictionary (takes X and Y points, takes tile size and bounds)

mapdata = string array of Level Text file

mapX = length of mapdata [0]

mapY = length of mapdata

(creates a grid of the map data in the array)

mapSize = point(mapX,mapY)

x, y = 0

while y < mapY:

newTiles = mapdata[y]

while x < mapX

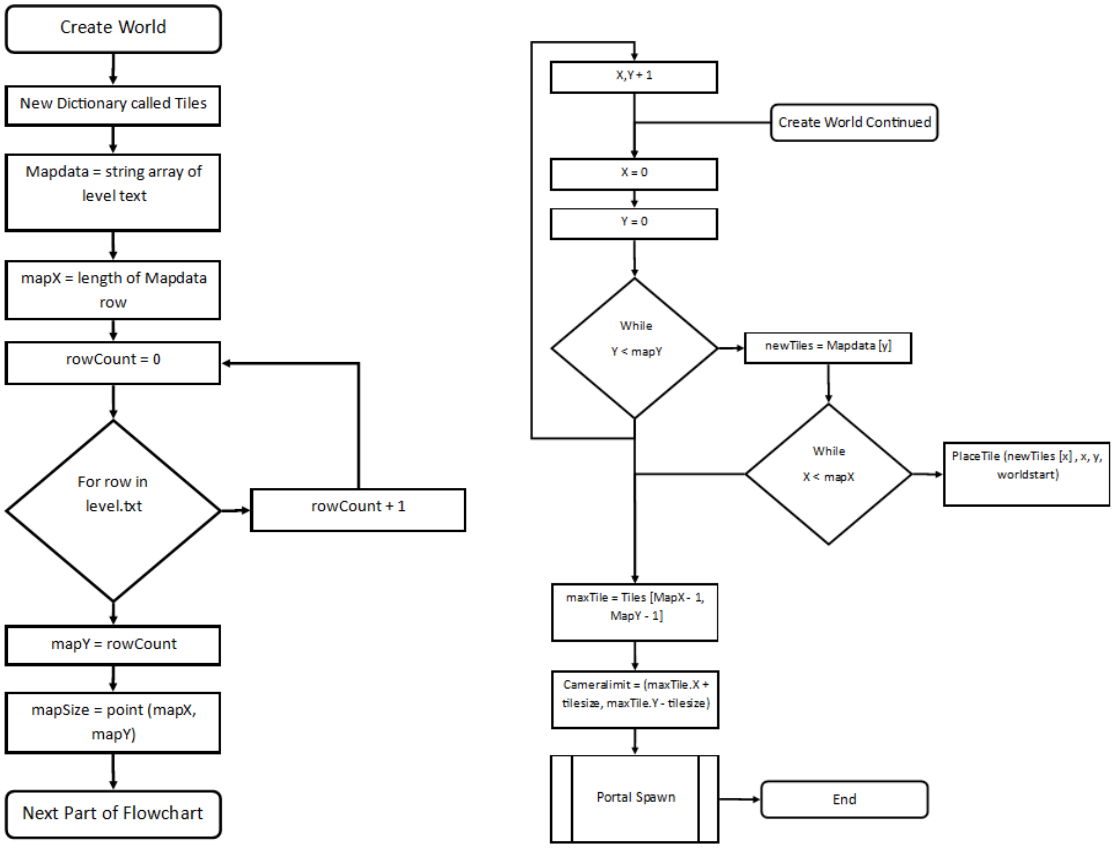
Place Tile(newtiles[x], x, y, worldstart)

x, y increment by 1

maxTile = Tiles[mapX - 1, mapY - 1]

cameralimit = (maxTile.X + tilesize, maxTile.Y - tilesize)

Function (Portal Spawn)



The modules above are used to generate the game world from a collection of tiles that are stored within the game, these tiles are generated if the index they are stored in is referenced in the level.txt file. The program then iterates through the columns and rows of the text file and instantiates tiles at the corresponding positions.

Private Function (Portal Spawn):

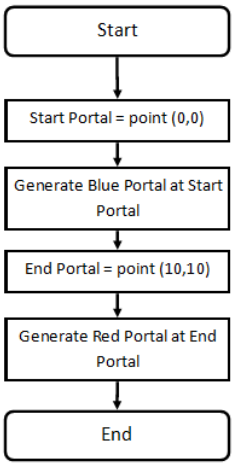
StartPortal = point (0,0)

generate BluePortal at StartPortal

EndPortal = point (10,10)

generate RedPortal at EndPortal

This module spawns the start and end point for the game, this is where enemies will spawn at, path towards and despawn at. The positions are currently set but can be changed at any time.



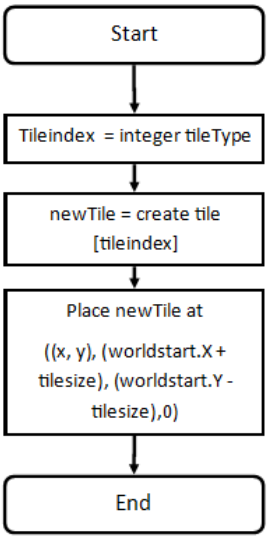
Private Function (Place Tile(tileType, x, y, worldstart)):

tileIndex = int (tileType)

newtile = create(tile[tileIndex])

place newtile at (point(x, y), (worldstart.X + tilesize), (worldstart.Y - tilesize), 0)

This module is used for actually instantiating the tiles in their assigned grid positions in the game world.



Public Function (Generate Path)

path = AStar getpath (StartPortal, EndPortal)

Game Management (Game World)

Private Variable Currency = 5

Private Variable Wave = 0

Private variable Lives = 10

Private Boolean gameOver = false

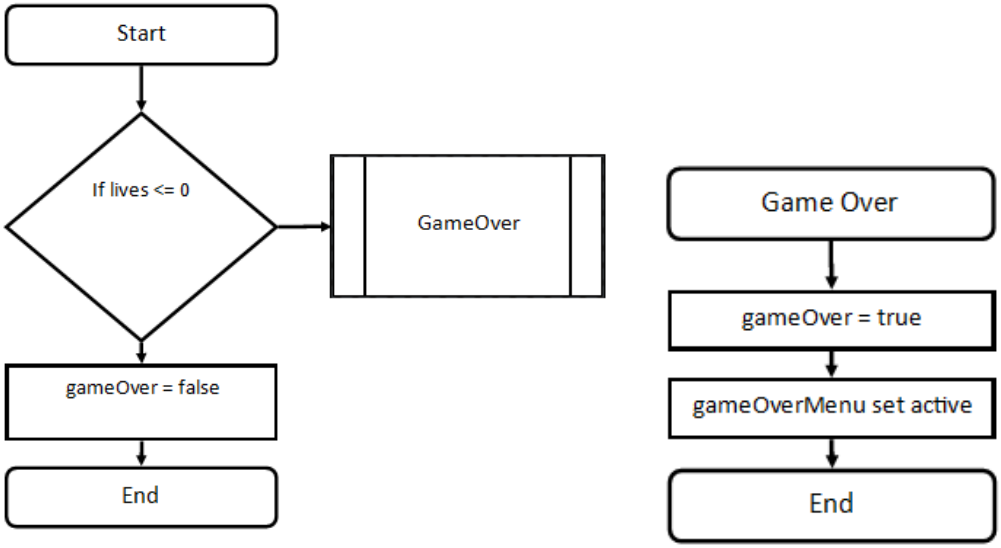
Public Function (Lives):

if lives <= 0

Function (Game Over)

else:

gameOver = false



Public Function (GameOver):

gameOver = true

gameOverMenu set active

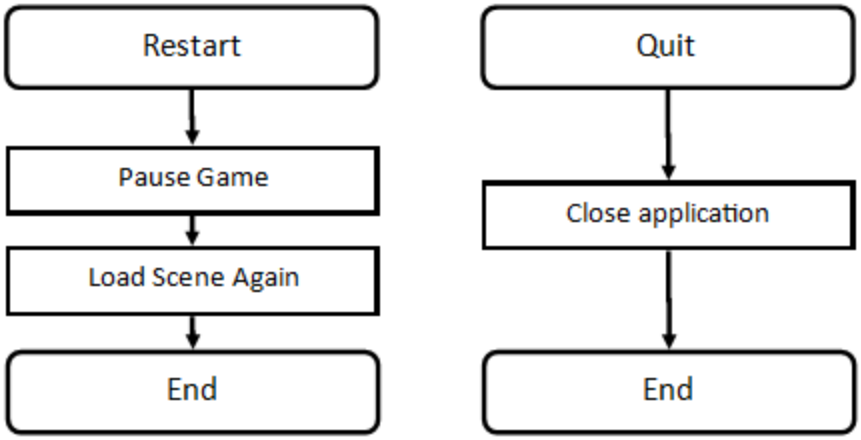
Public Function (Restart):

Pause Game

Load Scene

Public Function (Quit):

Close application



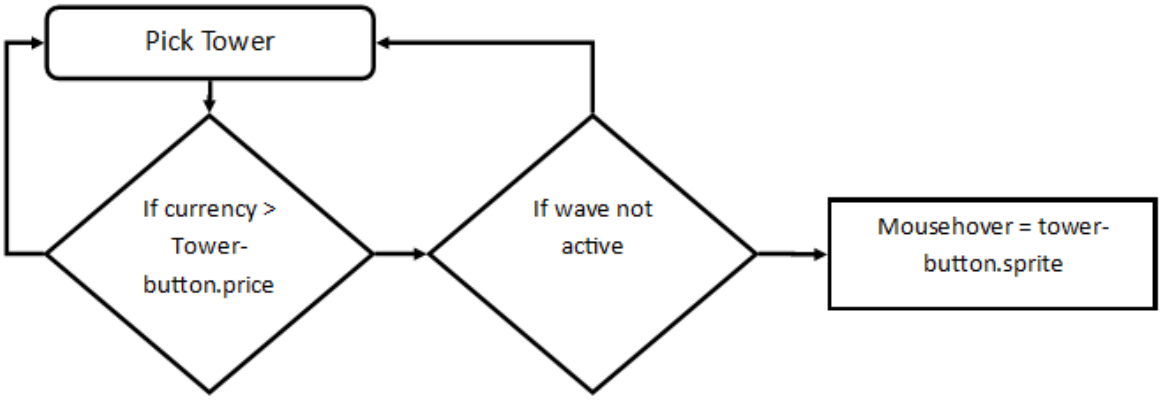
Public Function (Pick Tower(TowerButton):

if Currency >= TowerButton.price and WaveNotActive

ClickedButton = TowerButton

MouseHover = TowerButton.Sprite

This module allows the user to select a tower, if they cannot afford the tower it cannot be selected.



Public Function (BuyTower):

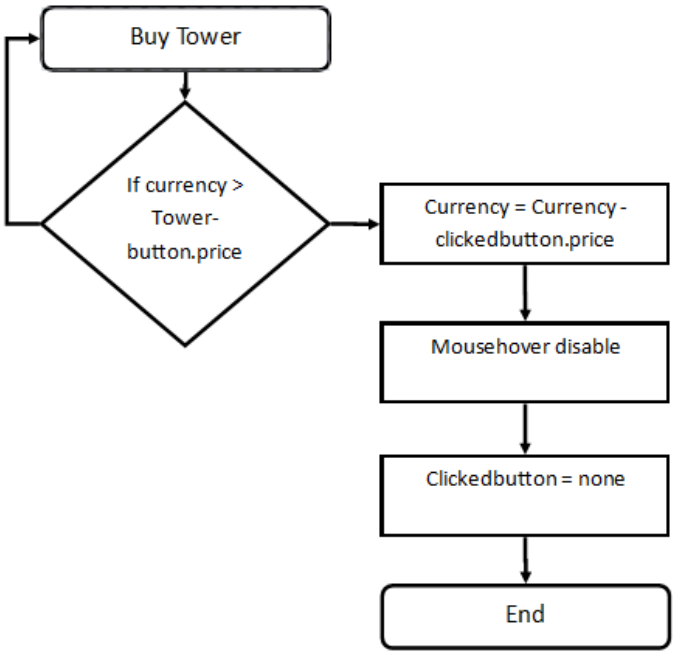
if Currency >= ClickedButton.price

Currency -= ClickedButton.price

MouseHover disable

ClickedButton = none

The module above is used to decrease the players currency by the cost of the tower they just placed.

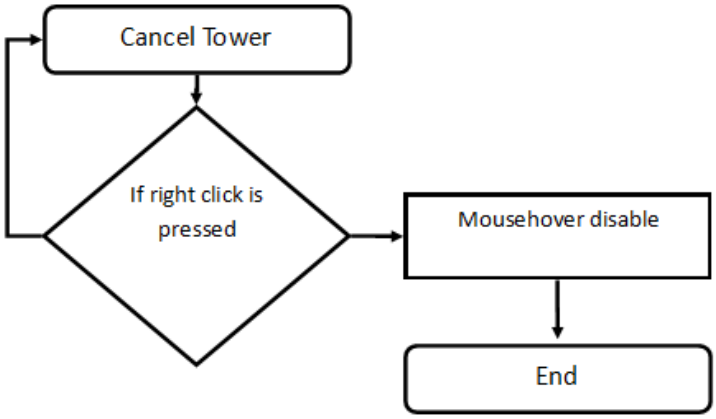


Private Function (Cancel Tower):

If input = right click

MouseHover disable

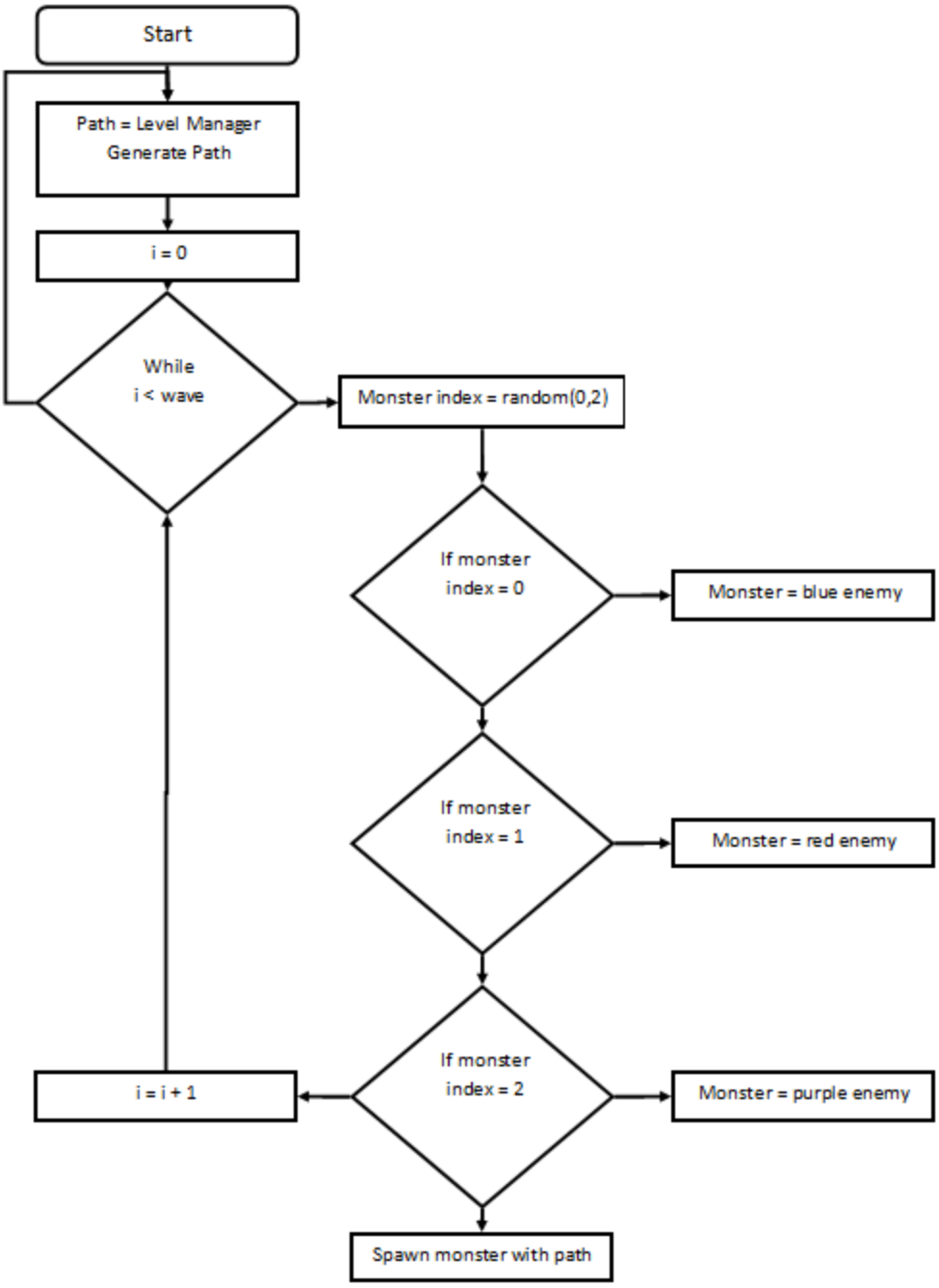
This allows the user to cancel the tower they have selected.



Private Function (Spawn Wave):

```
path = LevelManager.GeneratePath
i = 0
while i < wave:
    monsterIndex = random(0,2)
    if monsterIndex = 0
        monster = enemyBlue
    if monsterIndex = 1
        monster = enemyRed
    if monsterIndex = 2
        monster = enemyPurple
    Spawn monster with path
```

This module spawns an amount of enemies equal to the wave number and spawns random enemies from an array of enemy sprites.

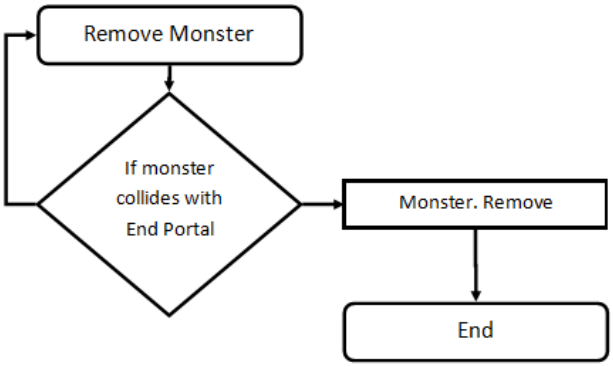


Public Function (Remove Monster):

if monster collides with EndPortal

monster.remove

Lives -= 1



Tile Management (Tile Script)

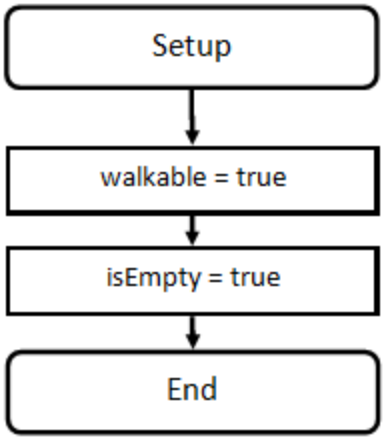
Public Function (Setup)

Walkable = true

isEmpty = true

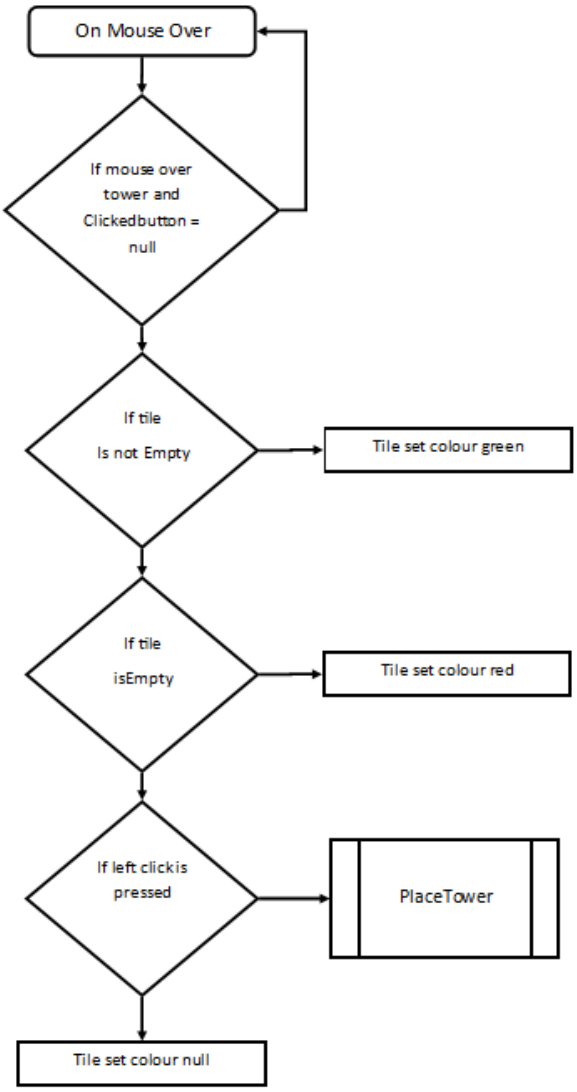
FreeTile = colour(Green),transparency = 50%

FullTile = colour(Red),transparency = 50%



```
Private Function (OnMouseOver)
        if mouse over gameObject and ClickedButton = null
                if tile isEmpty
                        Tile.setColour(FreeTile)
                if tile !isEmpty
                        Tile.setColour(FullTile)
                else if mouseInput = left click
                        Function (PlaceTower)

Private Function (OnMouseExit)
        Tile.setColour(null)
```

The two modules above change the colour of tiles to show their availability for a tower to be placed on them.

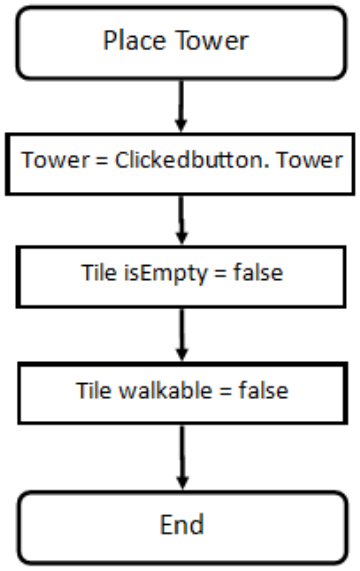


Private Function (PlaceTower)

tower = create GameObject ClickedButton.Tower

Tile IsEmpty = False

Tile Walkable = False



AStar Algorithm (Complexity of the Project)

```
Private Dictionary nodes (Point, Node)
Private Function (CreateNodes)
        nodes = new Dictionary entry
        for each tile in LevelManager
                nodes.add (tile.position, new node)
Public Stack Node GetPath (Start, Goal)
        if nodes = null
                Function (CreateNodes)
        New Stack openlist
        New Stack closedlist
        CurrentNode = nodes [Start]
        openlist append (CurrentNode)
        While openlist.Count > 0
                x = -1
                y = -1
```

```
        while x <= 1
            while y <= 1
                neighbour = (CurrentNode.X - x, CurrentNode.Y - y)
                if neighbour InMapBounds and walkable = true and != currentNode
                    gCost = 0
                    if x - y = 1
                        gCost = 10
                    else
                        gCost = 14
                    nodes.add neighbour
                    if openlist contains neighbour
                        neighbour calc(currentNode, nodes[goal],gCost)
                    else if closedlist does not contain neighbour
                        openlist.add neighbour
                        neighbour calc(currentNode, nodes[goal],gCost)

        openlist.remove(currentNode)
        closedlist.add(currentNode)

        if openlist.count > 0
            currentNode = openlist order by value calculated
        if currentNode = nodes[goal]
            while currentNode != start
                finalPath.Push(currentNode)

    return (finalPath)
```

The above module is my A* and it works be analysing each adjacent tile to the current tile to assign a value for it (from proximity to end point), it then compiles and compares a list of these

values and selects the smallest value each time, after this it then sets the smallest value as the next current tile and repeats this until the end point is part of the path.

Usability features and Data Flow

When creating my program I would like to ensure that I implement enough features to ensure easy navigation of the software for the user, this will hopefully include several features which optimise usability and and enhance the experience for the user.

To start I plan to create a simple GUI with a large enough scale that it is easy to understand and to see all of the features it provides, whist not being too large that it conceals too much of the game world. This will allow the user to easily find and use the features of the game and to reduce the uncertainty that a complex GUI would provide. I will do this by have clear sections on my GUI for each aspect; across the top of the application there will be a large button for starting the next wave, this will also have the users in game money, life points and the current wave number. Down the right of the screen will be the towers the user can purchase, they will vary in price which will be easily identifiable. Each tower will hopefully have their own stats, meaning more expensive towers will apply more damage and potentially other effects such as slows, damage over time and area of effect damage.

Another feature which will affect usability is camera movement, this feature will allow the user to navigate around the game world using the controls WASD in the same sense the arrow keys work. I have chosen to use these four keys as most video games on computers these days follow a standard format for controls and movement is usually assigned to these keys. This will provide users with a similar feel to the game so they can pick it up faster. The camera movement will also be useful as the software can run on different devices, this means that smaller monitors will see less of the game world and the camera movement removes the visual implications this would have.

More features which impact the usability of the application are the labels and titles given to certain aspects of the GUI, for example the cost of towers and the currency are both clearly displayed on the GUI and this allows for an easy way for the user to monitor their expenses in the game.

The main changes to data within the game is the currency, this is affected when the user buys something or when the user receives a reward for destroying an enemy. The game will then use this data collected throughout the user's experience to calculate a score, it will do this by calculating the total amount of money gained by the user during the game and it will multiply it by the wave number. This highscore will then be displayed on the "Game Over" menu.

These scores will be stored externally in a text file saved in the same directory as the softwares source files, and if possible due to time constraints these scores will be accessed through the

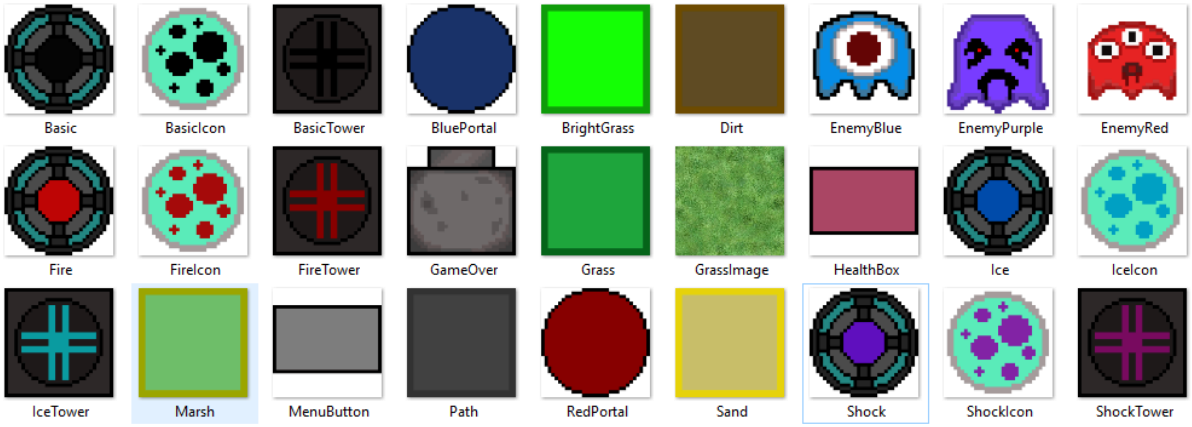
game, sorted using the quicksort algorithm into highest to lowest order and then the top 3 scores will be displayed through the game to the users as the highscores.

Sprite Designs

Before creating my software I would have attempted several different approaches for sprites in my game, this ranges from the types of tiles the world will consist of, the portals the enemies spawn and despawn at and the towers that will be placed in the world.

Below is a collection of designs for sprites, icons and GUI features that I have created and plan to use in my game. Not all of these designs will be used and some may be used as placeholders until I update the sprite or change the design completely.

The sprites, while essential to make the game appear visually pleasing, are not the highest priority for the software I plan to create. My aim is to focus solely on the complexity of code I can accomplish and to see how well I can apply the workings of an A* algorithm to a game that can utilise its pathing techniques.



Variables, Data Structures and Validation

Module	Variable Name	Data Type	Stores
SetLimits	xMax	float	maximum x position
	yMax	float	maximum y position
TileScript	IsEmpty	boolean	if a tile is empty
	fullcolor	color32	colour overlay for tile
	emptycolor	color32	colour overlay for tile
	Walkable	boolean	if a tile is passible
Tower Button	price	integer	tower cost
	priceTxt	string	tower cost
AStar	gCost	integer	node cost
	ConnectedDiagonally	boolean	how tile is connected
ClickedBtn	currency	integer	users currency
	wave	integer	wave number
	lives	integer	amount of health
	gameOver	boolean	if game is finished
	livesTxt	string	amount of health
	CurrencyTxt	string	users currency
	WaveText	string	wave number
	WaveActive	boolean	if wave is active
	Currency	integer	users currency

LevelManager	TileSize	float	size of tile
	blueSpawn	point	position for bluespawn
	redSpawn	point	position for respawn
	mapSize	point	size of map
	path	stack	stack for A*
	Path	stack	stack for A*
Monster	speed	float	monster speed
	progress	float	for despawn scaling

Key Data Structures

While creating my program I will not only use variables, I will make use of lists, arrays, dictionaries and stacks. These will be used to reduce the amount of programming that must be done and will work to store similar game objects and features together, when creating an A* algorithm the easiest way to manage the nodes that the sprite will travel across is by calculating them and storing them in a stack, this is because they will be calculated in the order they will be travelled and as stacks work via a first in first out system they will work very effectively at storing my node path.

I will also use arrays to store my sprites within my game, as there will be multiple towers and tiles which will be used throughout the game, storing them in image arrays within the level and game managers will make them easier to manage, modify and reference within the code.

Another vital data structure for my code is the text file that will store map data, this will be used to store the map structure which the game world will be generated from. Storing this file externally allows for easy modification and can have the potential for multiple maps that can be loaded from the same file. This method for generating my map will reduce the coding required to generate it significantly.

I also plan to use a CSV file to store the users score when they have completed the game, this gives me an easy method to conveniently access scores which will be clearly separated. As they are stored in this way it will make sorting the scores in ascending order less challenging and means a more complex high score system could be possible if the time constraints are not an issue.

Key Modules and Classes

My program will function around two main constructs, this will include the Level Manager and the Game Manager. The level manager will be responsible for generating the map, calculating map limits and camera movement functionality, it will be in charge of generating the start and end portals for the game world for the A* to use and will contain dictionaries of the tiles used by the level text file. This module will control where the camera starts, its limits and the speed and controls used by it, these will all be defined in subroutines referenced and organised by the level manager.

The game manager controls the in game experience, it spawns the enemies and applies the A* path to them. This module maintains and manages the in game variables such as waves, currency and lives, ensuring that they are updating during any events in game. This module also controls the restarting and closing of the game when the user loses, in this case they are given a menu with the choice to restart or quit which will be controlled by the game manager. The game manager also ensures the user can purchase towers and place them in the game world, preventing towers to stack on the same tile and updates their in game balance after purchasing a tower. This module runs the wave generation and allows enemies to randomly spawn from a sprite dictionary stored in the game manager.

Validation

When creating a game validation is important to ensure accuracy and consistency throughout the users experience. This means that I will need to ensure simple features cannot be used in a way that it will affect the experience negatively. i plan to ensure the user is not allowed to place multiple towers on the same tiles as this could cause entity lag and potential issues when towers encounter enemies. To combat this i will put in place variables that remove the ability for towers to be placed on a tile if it already contains a tower, this will be visually displayed to the user by a change in the colour of the tile. When placing a tower the tiles below the cursor will be highlighted respectively depending on if it is allowed to be placed there, green for available and red for not available.

Another issue that must be validated is the user's currency throughout the game, this will need to be updated throughout the game to ensure the user does not get towers for free or does not receive money when an enemy is destroyed. This will be done by linking the currency to a currencyTxt variable, these will translate the changes in currency in the code and will update the GUI's value simultaneously. The same precautions will be taken with the users HP as the game can rely completely on whether the user has full health or 1 health, this will ensure they can play strategically without the game inhibiting their maximum potential.

The buttons used in the GUI of the game will also need to be validated, this will ensure the user cannot press certain buttons at phases through the game. For example I would like to ensure

the user can only purchase and place towers between waves, not during as this could lead to issues with the A* having to update its path while sprites are moving. To avoid this the tower purchasing buttons will be disabled during waves. Another concern is the next wave button being used while a wave is already in progress, this could cause issues with the incorrect amount of enemies spawning in a certain time frame and will be avoided by disabling the next wave button while a wave is in progress.

Test Strategy and Research

The development process of my my application will only be complete once all code has been tested, this will provide me with a measurement of how functional the program is and will help me identify issues that must be resolved. To ensure my software is as good as I can make it I will be using multiple test strategies to verify each major function.

Camera Movement

Test No.	What is tested?	How is it tested?	Why it is tested?	Test Data			Expected Result
				Normal	Abnormal	Extreme	
1	WASD controls for camera movement	Visual monitoring of the game when these keys are pressed	Cannot navigate game world if not functional	W	F	/	Camera will pan across the game world
1.1	Limits of camera against map border	moving camera to edge of map	Camera should be locked only to game	Moving camera within game world	Moving camera near edges of game world	Moving camera only against edges and corners	Camera should stay locked to map limits

Tower Purchasing and Currency

Test No.	What is tested?	How is it tested?	Why it is tested?	Test Data			Expected Result
				Normal	Abnormal	Extreme	
2	If tower button is clicked, tower attaches to mouse	clicking a tower button	To show the user which tower they have clicked	N/A	N/A	N/A	Tower will attack an icon to the mouse
2.1	if right clicking when a tower is selected cancels that tower	Selecting a tower then right clicking	Gives user option to change their mind	N/A	N/A	N/A	Tower icon removed from cursor
2.2	If tower is placed with left click	Placing a tower down	To check currency is	N/A	N/A	N/A	Currency decreases by

	currency is changed	with left click	updated in real time				tower cost
2.3	Towers cannot be placed on top of each other	Attempt to place two towers on same tile	To ensure towers cannot stack	N/A	N/A	N/A	Tile will highlight red and will not place
2.4	Towers can only be placed when they can afford the tower	By trying to buy a tower that costs more than currency	To prevent users from getting towers for free	N/A	N/A	N/A	Will not be able to purchase tower

A* Navigation

Test No.	What is tested?	How is it tested?	Why it is tested?	Test Data			Expected Result
				Normal	Abnormal	Extreme	
3	Start node and end node are identified	Using a debugger that highlights start and end	to check the path starts and ends at correct points	N/A	N/A	N/A	They will both be highlighted green and red respectively
3.1	Pathway is shortest path available	Recreate the map in a known A* and compare paths	To make sure the complexity of the program is accurate	N/A	N/A	N/A	Shortest path will be calculated
3.2	Objects such as towers are not included in path	Debugger shows pathway not including towers	To prevent monsters colliding through towers	N/A	N/A	N/A	Towers will be avoided by enemies
3.3	Cannot create diagonal connections across blocked tiles	Debugger shows path avoids diagonal connections near unwalkable tiles	To prevent monsters colliding through towers	N/A	N/A	N/A	Towers will not be cut across by enemies
3.4	Path is updated between waves	Path is observed before and after changes are made between waves	To make sure monsters path around any changes made such as new towers	N/A	N/A	N/A	Path is recalculated to stop monsters crossing over new towers
3.5	Path is applied equally to all	Check to see if all enemies	To see if all monsters	N/A	N/A	N/A	Enemies will all follow the

	enemies that spawn	follow same path	follow the same path				same path

Monster Spawning and Pathing

Test No.	What is tested?	How is it tested?	Why it is tested?	Test Data			Expected Result
				Normal	Abnormal	Extreme	
4	Monsters spawn randomly	Check if there is use of random integers in code for spawning	To add some variety to the waves	N/A	N/A	N/A	Monsters spawn randomly
4.1	Monsters spawn at blue portal	See if monsters appear from blue portal	Because enemies should only spawn from the start point	N/A	N/A	N/A	Monsters spawn at blue portal
4.2	Monsters despawn at red portal	See if monsters disappear at red portal	To check that monsters are removed at the end	N/A	N/A	N/A	Monsters disappear at red portal
4.3	Monsters despawn when destroyed	Observe whether monsters disappear when attacked by tower	To make sure monsters do not stay after being destroyed	N/A	N/A	N/A	Monsters disappear when destroyed by tower
4.4	Monsters are reused to prevent lag	Use a collider to find game objects and observe if it increases when no new monsters are observed	To reduce processing power that could be spent on other aspects of the game.	N/A	N/A	N/A	Monsters will be reused
4.5	Monsters are not spawned on top of each other	Visually observe when monsters spawn	To stop monsters from overlapping as it reduces the quality of	N/A	N/A	N/A	Monsters wait between spawns to prevent overlap

			the game				

Tower Shooting

Test No.	What is tested?	How is it tested?	Why it is tested?	Test Data			Expected Result
				Normal	Abnormal	Extreme	
5	Tower shoots projectile at monster	See if projectile is fired at monster in adjacent tile	To make sure towers attack enemies	N/A	N/A	N/A	Tower shoots at monsters
5.1	Tower only fires projectile in certain radius	Place towers at different distances and see if all attack	To see whether towers range works correctly	N/A	N/A	N/A	Towers only shoot in their range
5.2	Tower fires set amount of projectiles per minute	Count how many times tower fires projectile	To balance the game and to make sure it isn't too easy	N/A	N/A	N/A	Towers only fire 20 times in a minute maximum
5.3	Projectiles are removed after fired	Observe unity hierarchy and see if they are removed	To prevent lag as too many entities can cause performance issues	N/A	N/A	N/A	Projectiles are removed after they are fired

New Waves and Between Waves

Test No.	What is tested?	How is it tested?	Why it is tested?	Test Data			Expected Result
				Normal	Abnormal	Extreme	
6	Next Wave button works	See if enemies are spawned when pressed	To make sure another wave can start	N/A	N/A	N/A	New wave will spawn when pressed

6.1	Towers cannot be placed between waves	Try to place a tower between waves	To prevent issues with A*	N/A	N/A	N/A	Towers will not be able to be placed
6.2	Enemies spawned is equal to wave number	Observe amount of enemies per wave and compare	Because it is a game feature	N/A	N/A	N/A	Enemies spawned will be equal to wave number

Health Points

Test No.	What is tested?	How is it tested?	Why it is tested?	Test Data			Expected Result
				Normal	Abnormal	Extreme	
7	Health points decrease when enemies reach end portal	Observe if health points decrease when enemies reach end point	Because it is a game feature and the user must attempt to prevent this	N/A	N/A	N/A	Health points will decrease when enemy reaches end portal
7.1	Always start with 10 health points	Check at the start of the game	To make sure all users always have a fair start	N/A	N/A	N/A	User will always start with 10 HP
7.2	Game Over menu appears when health points reaches 0	See if menu appears when HP reaches 0	To allow the user to restart or quit the game	N/A	N/A	N/A	Game over menu will appear at HP=0

Post Development

For my post development testing I will have a series of documents and forms I will provide to my stakeholders to gather feedback on the software based on the experience they have with it. The forms I will be using will be similar to the examples below:

Module	Rating (1 - Lowest 5-Highest)	Feedback

	Reason Optional	
Camera Movement		Positive:
		Negative:
		Improvements:
Tower Purchasing and Currency		Positive:
		Negative:
		Improvements:
A* Navigation		Positive:
		Negative:
		Improvements:
Monster Spawning		Positive:

		Negative:
		Improvements:
Tower Shooting		Positive:
		Negative:
		Improvements:
New Waves / Between Waves		Positive:
		Negative:
		Improvements:
Health Points		Positive:
		Negative:
		Improvements:

Testing Modules against Success Criteria

Primary Goal	Criteria for goal	Criteria met/not met	Primary Goal met/not met
Allowing the User to access a settings menu from the game	Settings Menu can be accessed in the game		
	User can change sound settings		
	User can change difficulty settings		
GUI of the application is easy to understand and an explanation is available to the user in game	User can access in game explanation		
	User can comfortable use and navigate the GUI		
	GUI will be self explanatory and will not require explanation		
Application provides fun and challenging experience and is balanced	Game is balanced and fair making it completable		
	Application is challenging and self learning		
	Application has depth and provides an enjoyable experience		
Users score will be stored in a data file which can be sorted and return the top 3 scores	Users score is stored after completing the game		
	User can access and		

	view other scores in the data file		
	User can view the highscores people have achieved		
User can access a variety of towers within the game	User can purchase and place towers on the map		
	User can place multiple different towers on the map		
	User has access to multiple towers with different features		

Stakeholders Involvement:

For my post development testing scheme (shown above) I will be involving several stakeholders to separately analyse and identify the level of success each module has had in the final product. For this I will have two of my main stakeholders; Robert Allan and Daniel Rigby test the functionality of the in game experience, this will include tower placements, pathing algorithms, currency and lives, enemy spawning and despawning as well as the functionality behind the towers to destroy enemies within a range.

After a brief discussion with these stakeholders we have concluded that the success criteria accurately covers a wide variety of levels of success and can easily be used to measure the success and functionality of all modules determines in the initial requirements agreed.

Other stakeholders who will be involved with the post development testing are George Bond and James Foggin. These stakeholders will be used to analyse the efficiency of the code (precautions for potential lag issues) as well as map generation and level resetting. This is because they are both experienced programmers and avid gamers who understand common constructs within video games.

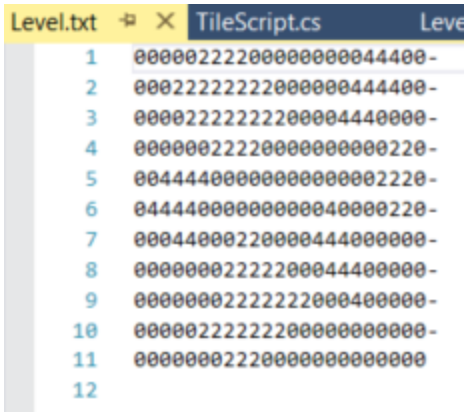
Section 3

Core Modules with Testing

For my project I have decided to program and execute my application using the Unity Engine, this means I will be programming in the programming language C#. Whilst this is a new programming language to me and it may be a challenge to begin learning I have some background knowledge of the basics and some experience with modular programming from my computer science GCSE and A level.

Level Manager

My first module of code is my level manager, this is the foundation for the beginning of my game as it controls the map generation. This module is used to read the level.txt file and to interpret the types of tiles which will be used and the positions they will be used in. The collection of these tiles will create a grid of terrain which will be the basic construct of my map. Lots of the features used in the level manager are more easily accessible when viewed from the inspector so I used a command that allows me to edit variables from the inspector.

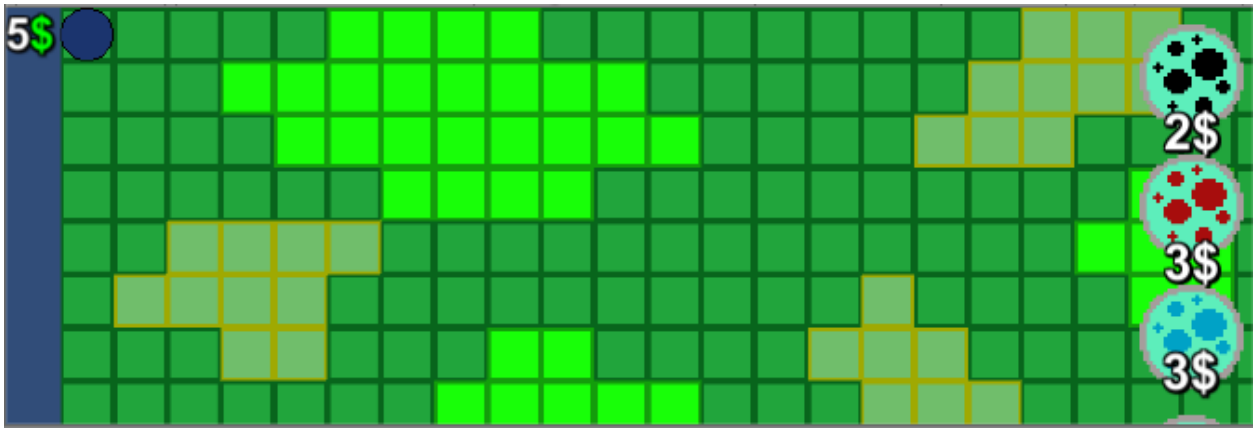


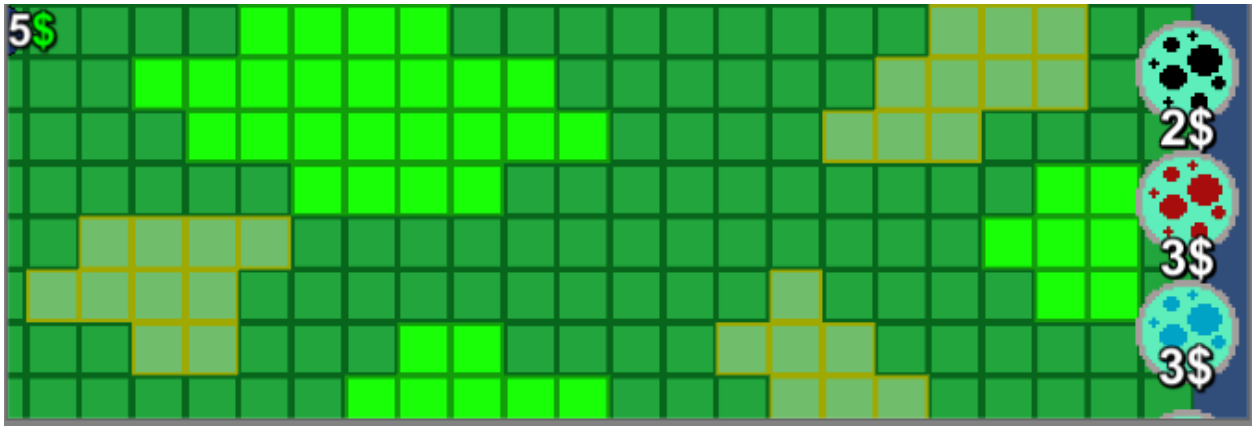
The above screenshot shows my Level.txt file, this will be the map that is generated and each number represents a tile from an image array in the levelmanager, this allows me to easily change how the map looks by changing the numbers around as long as they stay within the range of the array.

During the development process this level file was initially designed using only two of the tiles from the tile prefab array, I thought this looked a little bland and decided to use another of the tiles to add some variety to the map. The other tile prefabs are stored for

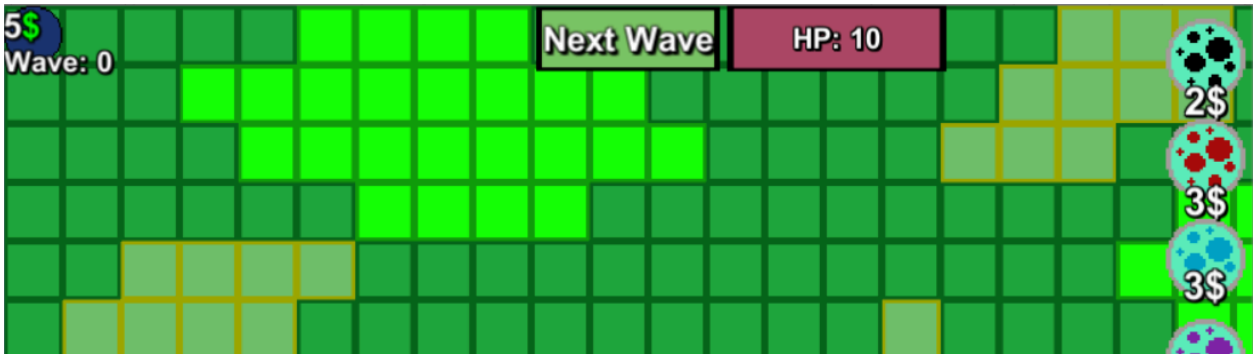
any potential other maps which the game could load in randomly. This also provides opportunity for multiple levels.

To test this module I ran the program on different resolutions, this would test if my map limits function would work. Initially it was programmed for a 1920 x 1080 resolution however when I ran it on 1176 x 664 resolution the blue background appeared near the borders.





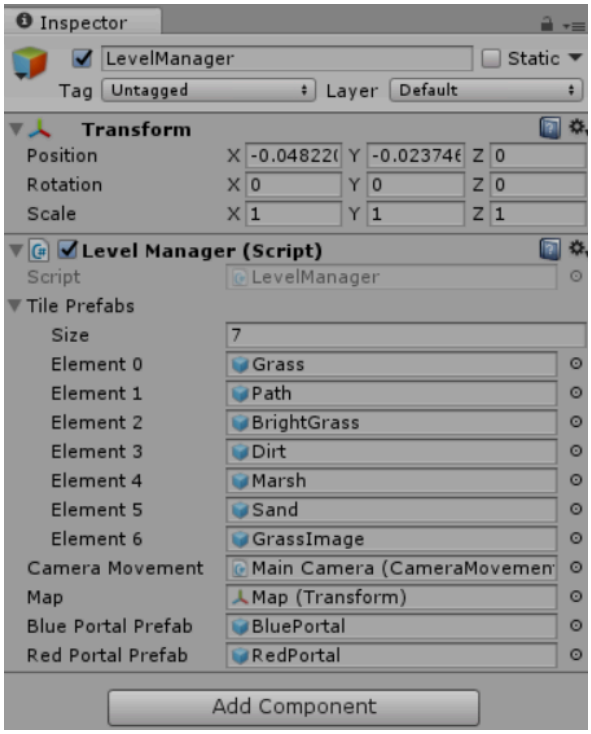
This was only an issue horizontally as the vertical borders were keeping the camera in the limits. To resolve this issue I had to change the centre of my tiles to top left rather than top right as it was acting as though another column of tiles should be there. I also noticed some of my buttons from my UI were missing so I had to modify the layer order they appear in to make them appear on different resolutions.



This screenshot shows the game after resolving UI issues and map limits for the camera.

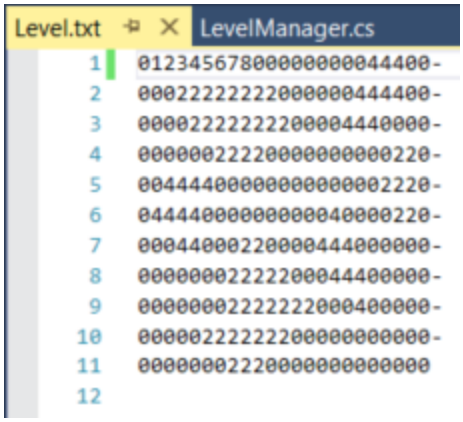
```
using System.Collections;
using System.Collections.Generic; // used for tile dictionary
using UnityEngine;
using System;


public class LevelManager : Singleton<LevelManager>
{
    // Used to show all features in inspector
    [SerializeField]
    // create array of tile types
    private GameObject[] tilePrefabs;
```

This block of code is the introduction to my level manager class, it demonstrates the libraries which are built into unity which I will be using. As I would like certain variables and constructs to be accessible from the unity editor I have used the command [SerializeField] to make them appear.



This image shows the multiple variables and the image array I have made accessible in the inspector and shows the collection of tile prefabs which will be used in making the game world. The position in the array of the tile prefab refers to the number on the level.txt file which is how tiles are referenced and accessed.

To test the array of tiles I had created I modified my lvl.txt file with a variety of numbers to show that all tiles appeared, I also tested indexes out of the range of the table. This returned some interesting results. All tiles in the table appeared as expected from the text file, however the values that were out of the range of the table causes an error and the game could not read the rest of the text file and just cut off where the first out of range value appeared.



The above screenshots show the changes I made to test the sprite array for my tiles and how it impacted the game after an out of range error occurred. I currently don't have a fix for this apart from ensuring the predetermined level file is all in range, however for the future when planning on procedurally created maps I must ensure it has limits to spawn random tiles from to prevent this issue.

The next portion of my level manager shows more variables and constructs which are used in the level manager, I have stayed to a format of keeping variables and other data types within the top of my classes.

```
// Gets the size of each tile (all same resolution)
public float TileSize
{
    // returns the first tile in the prefab folder using sprite render and sprite bounds
    get { return tilePrefabs[0].GetComponent<SpriteRenderer>().sprite.bounds.size.x; }
}

// Used to show all features in inspector
[SerializeField]
// make local variable called cameraMovement
private CameraMovement cameraMovement;
// Create global dictionary to get the tile type at a given point

[SerializeField]
private Transform map;

private Point blueSpawn, redSpawn;

[SerializeField]
private GameObject bluePortalPrefab;
[SerializeField]
private GameObject redPortalPrefab;


public Portal BluePortal { get; set; }

private Point mapSize;

private Stack<Node> path;
```

Next is the beginnings of where I link my A* algorithm into the program, this section of the level manager ensures a path has been calculated for the enemies to travel along and controls the generation of the map within the Start procedure. In this procedure I reference a function which will be run when the game starts called CreateLevel.

```
public Stack<Node> Path
{
    get
    {
        if (path == null)
        {
            GeneratePath();
        }
        // creates new stack of nodes for each individual monster, so monsters dont access and edit the same stack (error potential)
        return new Stack<Node>(new Stack<Node>(path));
    }
}

public Dictionary<Point, TileScript> Tiles { get; set; }

// Use this for initialization
void Start()
{
    // Starts game
    CreateLevel();
}
// Update is called once per frame
void Update() {
}
```

Create Level is a function that calculates the map limits for the camera to use and focuses on converting the text from the Level.txt file into a map that can be created. It reads the file and uses 2 for loops nested in each other to create the X and Y

dimensions for the map from the level file. Here I encountered an error where I the tiles I created in the game world were pivoting around the top left corner which shifted the tiles out of the map limits by half a tile up and out, this was fixed by manually changing the pivot of the tiles to the central point.

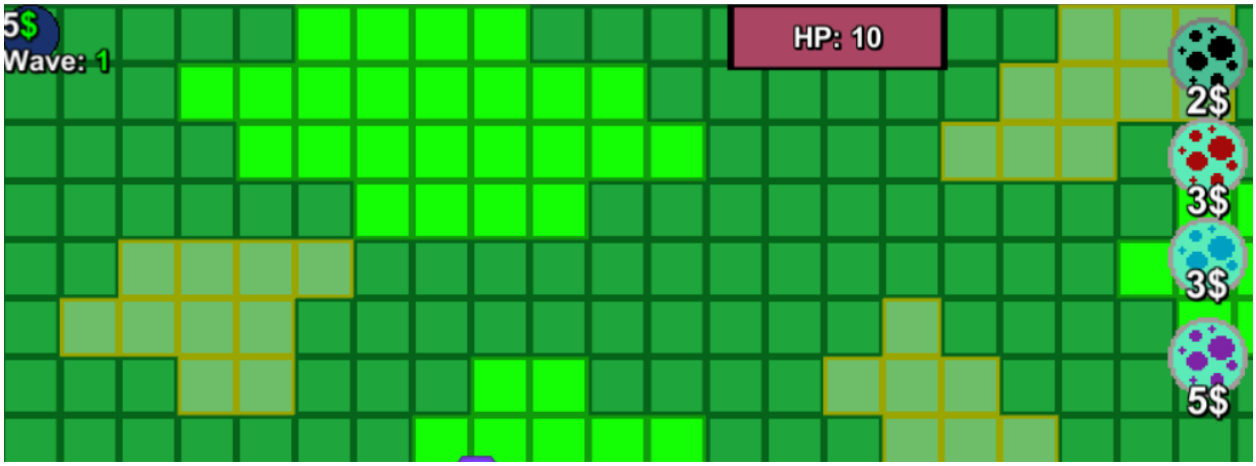
```
private void CreateLevel()
{


    // Stores new tile dictionary in variable Tiles
    Tiles = new Dictionary<Point, TileScript>();

    string[] mapData = ReadLevelText();
    int mapX = mapData[0].ToCharArray().Length; // creates rows of each tile with length of number of characters in that position in string array
    int mapY = mapData.Length;

    // x and y values for dimensions of the map
    mapSize = new Point(mapData[0].ToCharArray().Length, mapData.Length);

    Vector3 maxTile = Vector3.zero;
    Vector3 worldStart = Camera.main.ScreenToWorldPoint(new Vector3(0, Screen.height));

    for (int y = 0; y < mapY; y++)
    {
        char[] newTiles = mapData[y].ToCharArray();

        for (int x = 0; x < mapX; x++)
        {
            PlaceTile(newTiles[x].ToString(), x, y, worldStart);
        }
    }
    maxTile = Tiles[new Point(mapX - 1, mapY - 1)].transform.position;

    cameraMovement.SetLimits(new Vector3(maxTile.x + TileSize, maxTile.y - TileSize)); // Error here using just maxtile, had to add tile size on as i

    // After generating Map generate portals
    SpawnPortals();
}
```

This part of Level Manager also refers to a function called SpawnPortals which will generate the start and end portal of the game world. Next is the function PlaceTile which is used to actually place the tiles in the game world, it does this by selected the tile from the tile prefab array and instantiates it in the game world using the grid positions calculated from the nested For loops used before.

```
private void PlaceTile(string tileType, int x, int y, Vector3 worldStart)
{
    // Turns the tiletype into an integer so we can index
    int tileIndex = int.Parse(tileType);

    // Create new tile and reference to tile in newTile
    TileScript newTile = Instantiate(tilePrefabs[tileIndex]).GetComponent<TileScript>();

    // Assignes a grid position to the created tile
    newTile.Setup(new Point(x, y), new Vector3(worldStart.x + (TileSize * x), worldStart.y - (TileSize * y), 0),map);

    // ISSUE WHEN RUNNING BECAUSE SPRITE PIVOT WAS SET TO CENTRE AND NOT TOP LEFT SO HALF OF SPRITE WAS OFF SCREEN
}
```

When testing the A* I noticed that if a tower was placed in the game world while enemies were travelling to the end point they would move through the tower as the A* had not been updated that the tower tile was an object in the way. I could have made the A* run in the update function but that could have caused performance issues due to

the processing done, instead I made sure towers could not be placed while waves were active.



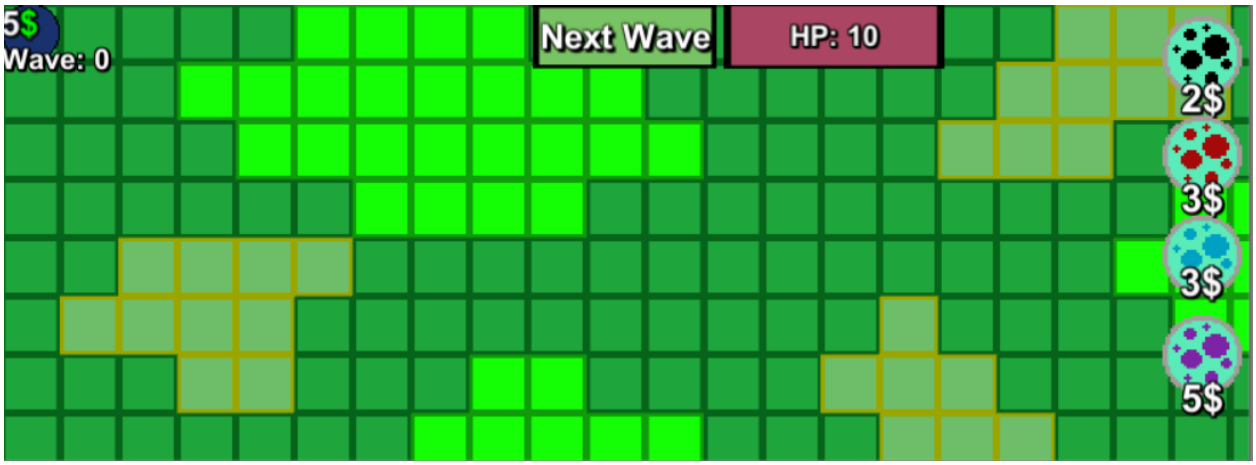
This screenshot shows that during a wave when a tower button is pressed (in this case the top tower button), the animation for it being pressed shows however a tower is not assigned to the cursor.

The final parts of the level manager controls the SpawnPortals procedure, the function which reads and converts the level.txt file into a usable format, the map limits and boundaries function and finally the A* GeneratePath procedure.

```
private string[] ReadLevelText()
{
    TextAsset bindData = Resources.Load("Level") as TextAsset;

    string data = bindData.text.Replace(Environment.NewLine, string.Empty);
    return data.Split('-');
}

private void SpawnPortals()
{
    blueSpawn = new Point(0, 0);
    // creates portal using the blue portal and by placing it in the centre of the tile which it found from the WorldPosition Function with no r
    GameObject tmp = (GameObject) Instantiate(bluePortalPrefab, Tiles[blueSpawn].GetComponent<TileScript>().WorldPosition, Quaternion.identity);
    BluePortal = tmp.GetComponent<Portal>();
    BluePortal.name = "BluePortal";


    redSpawn = new Point(14, 10);
    Instantiate(redPortalPrefab, Tiles[redSpawn].GetComponent<TileScript>().WorldPosition, Quaternion.identity);
}

public bool InBounds(Point position)
{
    return position.X >= 0 && position.Y >= 0 && position.X < mapSize.X && position.Y < mapSize.Y;

}

public void GeneratePath()
{
    path = AStar.GetPath(blueSpawn, redSpawn);
}
```

This section of code displays how the raw text from the level file is split by “-”’s in order to convert it into a grid format, making it generate the map from the grid created by splitting each line by the dash.

It also shows the grid locations of the blue and red portals which refer to the starting and end points respectively. These are also then instantiated into the game world at those grid coordinates. Lastly the GeneratePath function is used to call the A* algorithm written in another class to return a path which can be applied to monsters.



The above image demonstrates the game world and how it appears after all of the tiles have been generated, as you can see there is only one portal visible on the screen and that is the starting blue portal, the red portal is further down as the map is fairly large. After the camera has been moved down and to the right the end portal is in sight.

Camera Movement

I programmed this class in order to contain all of the camera movements and controls, it uses the update function built in to run procedure which checks for input from the user every tick, this allows me to check for the keyboard inputs from the user constantly and from this I attached the controls WASD to the camera movement. These keys act as

arrow keys would and move the camera around the screen the respective direction by a constant variable called cameraSpeed.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CameraMovement : MonoBehaviour
{
    [SerializeField] // allows you too see in the inspector
    private float cameraSpeed = 10;

    private float xMax;
    private float yMin;




    void Update()
    {
        GetInput(); // Call function every tick
    }
```

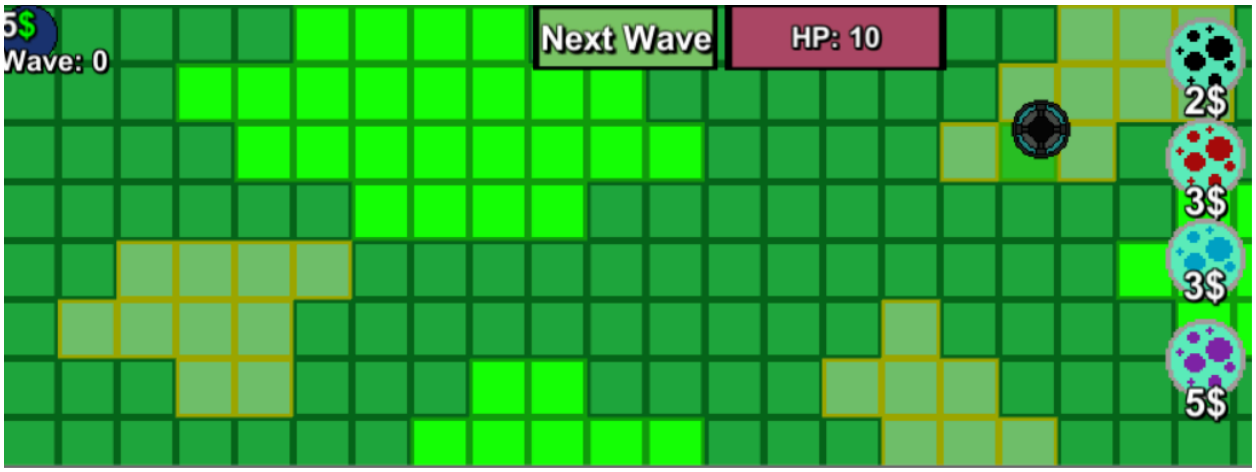
This screenshot shows the libraries from unity which are in use and the variables used in the class, cameraSpeed refers to the rate at which the camera will change in the X and Y dimensions when the camera movement keys are pressed.

```
private void GetInput()
{
    if (Input.GetKey(KeyCode.W))
    {
        transform.Translate(Vector3.up * cameraSpeed * Time.deltaTime); // time.delta time is the time since update was called
    }
    if (Input.GetKey(KeyCode.A))
    {
        transform.Translate(Vector3.left * cameraSpeed * Time.deltaTime);
    }
    if (Input.GetKey(KeyCode.S))
    {
        transform.Translate(Vector3.down * cameraSpeed * Time.deltaTime);
    }
    if (Input.GetKey(KeyCode.D))
    {
        transform.Translate(Vector3.right * cameraSpeed * Time.deltaTime);
    }

    transform.position = new Vector3(Mathf.Clamp(transform.position.x, 0, xMax), Mathf.Clamp(transform.position.y,yMin,0),-1);


}

public void SetLimits(Vector3 maxTile)
{
    Vector3 wp = Camera.main.ViewportToWorldPoint(new Vector3(1, 0));

    xMax = maxTile.x - wp.x; // gives distance camera can move
    yMin = maxTile.y - wp.y;
}
```

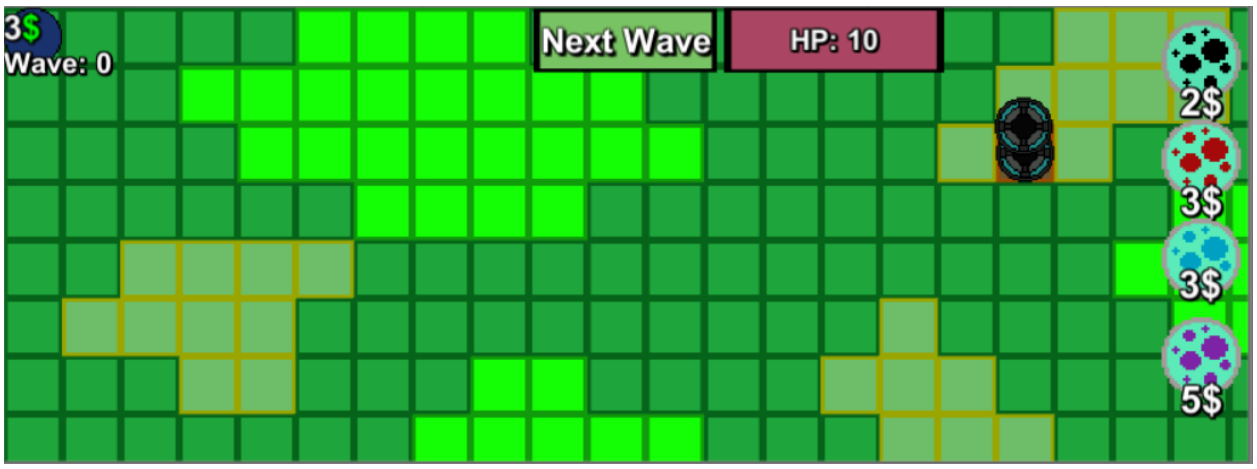
The above screenshot demonstrates how WASD are used to change the camera's position and the SetLimits functions is used to assign some maximum limits the camera can move to, this prevents the camera from moving outside of the game world as that could cause issues.

TileScript

This script is used for managing the individual tiles within the game, It keeps track of two main variables per tile, IsEmpty and Walkable. This refers to whether the tile has a gameobject placed on top of it, in which case it is no longer empty, and whether it is walkable. Walkable refers to whether the tile is included in the shortest path of the A*, unwalkable tiles are excluded and avoided. The function also applies effects to tiles and highlights them in different colours when a user is hovering over the tile with a tower selected. If a tile is not empty then it will highlight red to show the tile is not available for a tower, otherwise the tile will highlight green to indicate it is free.



In the top right of this image you can see the tower hovering over a tile and the tile below is highlighted green to show it is available.



This image shows the tower attempting to be placed on top of another tower. After the first tower was placed the tile was considered no longer empty and can no longer have game objects placed on top of it.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.EventSystems;

public class TileScript : MonoBehaviour
{

    public Point GridPosition { get; private set; }

    public bool IsEmpty { get; private set; }

    private Color32 fullColor = new Color32(255, 118, 118, 255); // red
    private Color32 emptyColor = new Color32(96, 255, 90, 255); // green

    private SpriteRenderer spriteRenderer;

    public bool WalkAble { get; set; }

    public bool Debugging { get; set; }

    // gets the centre point of a tile
    public Vector2 WorldPosition
    {
        get
        {
            return new Vector2(transform.position.x + (GetComponent<SpriteRenderer>().bounds.size.x / 2), transform.position.y - (GetComponent<SpriteRenderer>().bounds.size.y/2));
        }
    }
```

The image above shows the start of the TileScript class, this shows the variables which contain the colour overlays for tiles, the values for Walkable and IsEmpty and a line of code that finds the central point of a tile to ensure game objects placed on it are locked the centre.

```
//Use this for initialization
void Start()
{
    spriteRenderer = GetComponent<SpriteRenderer>();
}

//Update is called once per frame
void Update()
{

}

public void Setup(Point gridPos, Vector3 worldPos, Transform parent)
{
    WalkAble = true;
    IsEmpty = true;
    this.GridPosition = gridPos;
    transform.position = worldPos;
    transform.SetParent(parent);
    LevelManager.Instance.Tiles.Add(gridPos, this);
}
```

This image shows another portion of the TileScript, this mainly shows the Setup procedure which is in charge of defining the variables of a tile as the tile is generated.

As discussed previously this class is also in charge of changing the colour of the tiles below them to display whether they are available. This is done by the following two procedures.

```
private void OnMouseOver()
{

    if (!EventSystem.current.IsPointerOverGameObject() && GameManager.Instance.ClickedBtn != null)
    {
        if (IsEmpty && !Debugging)
        {
            colorTile(emptyColor);
        }
        if (!IsEmpty && !Debugging)
        {
            colorTile(fullColor);
        }
        else if (Input.GetMouseButtonDown(0))
        {
            PlaceTower();
        }
    }

}

private void OnMouseExit()
{
    if (!Debugging)
    {
        colorTile(Color.white);
    }

}
```

The image shows how the OnMouseOver procedure determines using a series of selection based queries, whether to highlight the tile with a certain colour, or whether the left click on the mouse is pressed. If this is the case it runs the PlaceTower function. Below that is the OnMouseExit procedure which simply resets the colour to its original when the cursor moves away from that tile.

Lastly in the TileScript class is the PlaceTower function, this function is in charge of placing the tower onto a tile and does so by instantiating the tower onto the tile and storing it as a child of that tile. The function makes sure the tile is on the top layer of the game so it doesn't clip behind anything, the layering feature was added as I encounter errors where the tower would be layered behind the tile it was placed on and would not be visible.

```
private void PlaceTower()
{

    GameObject tower = (GameObject)Instantiate(GameManager.Instance.ClickedBtn.TowerPrefab, transform.position, Quaternion.identity);
    tower.GetComponent<SpriteRenderer>().sortingOrder = GridPosition.Y;

    tower.transform.SetParent(transform); // makes tower placed child object of tile it was placed on

    IsEmpty = false;
    colorTile(Color.white);

    GameManager.Instance.BuyTower();
    WalkAble = false;
}

private void colorTile(Color newColor)
{
    spriteRenderer.color = newColor;
}
```

There is a clear demonstration of how the tiles are highlighted when tiles are empty and not empty further up the page.

Tower Buttons (GUI)

In the application I have created I have tried to provide a variety of towers for the user to use and place within the game world, I have done this by adding to the GUI a tower menu in which the user can select and place a tower they chose as long as they can afford it. This works with the use of buttons.

This image shows my canvas management and how I have structured and nested certain buttons and menus within the GUI. From here I can move and rescale the buttons and panels. I have encountered numerous scaling issues with the GUI here as I have tried ensuring the GUI will have the same scaling depending on the users monitor. After applying some pivot points to the game scene I have made the overlay stay in the same positions and spacing on 3 different resolution monitors I have tested.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class TowerBtn : MonoBehaviour {

    [SerializeField]
    private GameObject towerPrefab;

    [SerializeField]
    private Sprite sprite;

    [SerializeField]
    private int price;

    [SerializeField]
    private Text priceTxt;


    public GameObject TowerPrefab
    {
        get
        {
            return towerPrefab;
        }
    }

    public Sprite Sprite
    {
        get
        {
            return sprite;
        }
    }
```

```
public int Price
{
    get
    {
        return price;
    }
}

private void Start()
{
    priceTxt.text = Price + "$";
}
```

The above images shows the code for the tower buttons, it uses a series of Get and Return commands to return the tower the user has selected. From this I can instantiate them in the game world. Price is a function that updates the UI after the user has purchased a tower.

Monster

This class is all about the enemies in my game; it controls the amount of them that spawn, how they spawn, where they spawn, the speed they move at, the paths they take and lots of linking to other classes such as the game manager, the object pool (which I am using to reuse monsters to prevent lag) and the A* algorithm.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Monster : MonoBehaviour
{
    [SerializeField]
    private float speed;

    private Stack<Node> path;

    public Point GridPosition { get; set; }

    private Vector3 destination;

    private void Update()
    {
        Move();
    }
```

The class starts by importing the essential unity libraries as most of my classes do, then it contains the essential variables and data structures needed by the class. This includes the speed variable (which is accessible from the inspector as the field is serialised), as well as the stack which contains the path and the destination.

The image also shows a function being called within the update function, this means that every tick of the game the Move function is being called.

```
public void Spawn()
{
    transform.position = LevelManager.Instance.BluePortal.transform.position;

    StartCoroutine(Scale(new Vector3(0.1f, 0.1f), new Vector3(1, 1),false));

    SetPath(LevelManager.Instance.Path);
}

public IEnumerator Scale(Vector3 from, Vector3 to, bool remove)
{
    float progress = 0;

    while (progress <= 1)
    {
        transform.localScale = Vector3.Lerp(from, to, progress);
        progress += Time.deltaTime;
        yield return null;
    }
    transform.localScale = to;
    if (remove)
    {
        Release();
    }
}
```

This screenshot shows more of the Monster class and displays firstly, the Spawn function. This function sets the starting position as the blue portal, from here it them

scales the enemy monster sprite from nothing into its normal scale so it appears to come through the portal. Next the monster is then given its path by referencing the Path calculating from the Level Manager.

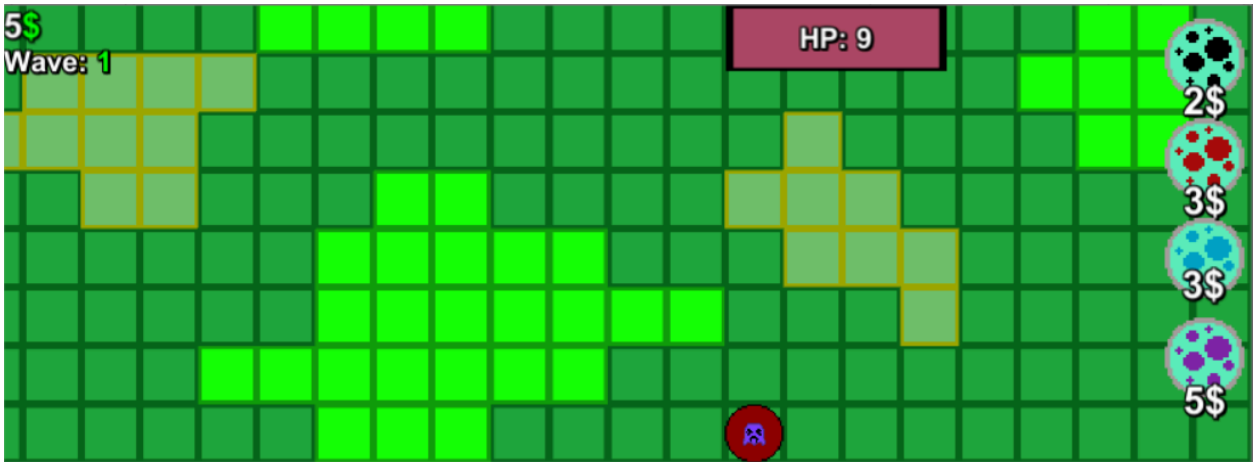
Next the screenshot shows the function that controls the scaling of the monster sprite, it works by using a progress variable which keeps track of the progress of the scaling of the monster. As the progress variable increases so does the scale of the monster sprite.

```
private void Move()
{
    transform.position = Vector2.MoveTowards(transform.position, destination, speed * Time.deltaTime);

    if (transform.position == destination)
    {
        if (path != null && path.Count > 0)
        {
            GridPosition = path.Peek().GridPosition;
            destination = path.Pop().WorldPosition;
        }
    }
}
private void SetPath(Stack<Node> newPath)
{
    if (newPath != null)
    {
        this.path = newPath;
        GridPosition = path.Peek().GridPosition;
        destination = path.Pop().WorldPosition;
    }
}
```

This screenshot above shows more of the Monster class, it covers two main functions; Move and SetPath. Move works by using the move towards command and uses a target location called destination which is previously defined as the red portal. The script shows how its next destination is defined as the next node within the A* algorithm, as it is a stack it can pop the front most node out of the list to set as its destination. SetPath is an alternative option which is used when the path hasn't changed, this means the monster can just pop off the node and use it as destination.

```
private void OnTriggerEnter2D(Collider2D other)
{
    if (other.tag == "RedPortal")
    {
        StartCoroutine(Scale(new Vector3(1, 1), new Vector3(0.1f, 0.1f),true));
        GameManager.Instance.Lives--;
    }
}

private void Release()
{
    GameManager.Instance.Pool.ReleaseObject(gameObject);
    GameManager.Instance.RemoveMonster(this);
}
```

The last two functions within Monster are used to despawn the enemies when they reach the redportal, they do this by using a collider. When the monster touches the collider it begins to scale down to nothing, when this happens the Lives variable is decreased as the player loses a HP for letting the monster reach this location. Release works simply to remove the gameObject from the scene however it keeps it in the object pool to reuse.



This image shows the enemy monster going through the red portal, as the monster is despawning it begins to scale down and as you can see across the top the HP as reduced by 1.

Hover

The next class I will be demonstrating is the hover class. The function of this class is to attach the tower icon to the cursor when a tower button is clicked. It uses a function called FollowMouse which uses the sprite renderer to move the tower icon by using an

input from the mouse which gives the mouse position on the screen.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Hover : Singleton<Hover>
{
    private SpriteRenderer spriteRenderer;


    // Use this for initialization
    void Start()
    {
        this.spriteRenderer = GetComponent<SpriteRenderer>();
    }

    // Update is called once per frame
    void Update()
    {
        FollowMouse();
    }

    private void FollowMouse()
    {
        if (spriteRenderer.enabled)
        {
            transform.position = Camera.main.ScreenToWorldPoint(Input.mousePosition);
            transform.position = new Vector3(transform.position.x, transform.position.y, 0);
        }
    }
```

The code above demonstrates how FollowMouse is called every tick of the game and it checks whether the button has been clicked, if it has then it moves the tower icon to the same position where the mouse is.

```
public void Activate(Sprite sprite)
{
    this.spriteRenderer.sprite = sprite;
    spriteRenderer.enabled = true;
}

public void Deactivate()
{
    spriteRenderer.enabled = false;
}
```

These 2 procedures work by giving two states to the icon attached to the mouse, activate and deactivate. All they do is enable and disable the sprite renderer for the tower icon when they are called.

Object Pool

The object pool class within my game is used to store monsters when they are not in the game scene, this could happen when they are spawned in one wave but not in the next, by storing them in an object pool I can reuse the same game object without having

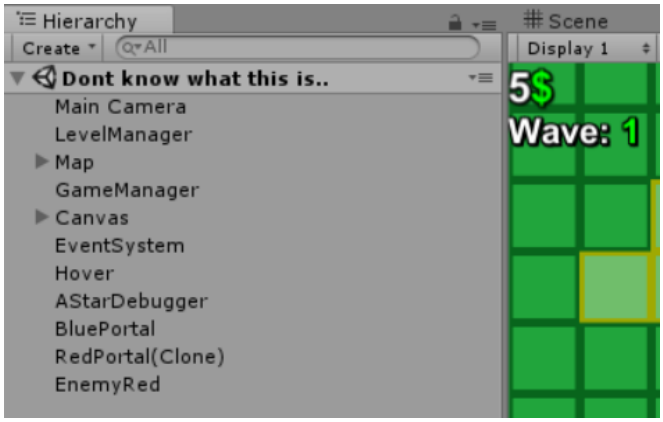
to create a new one. This reduces the processing power spent on that and also reduces the lag as there are less entities loaded in the game world at one time.

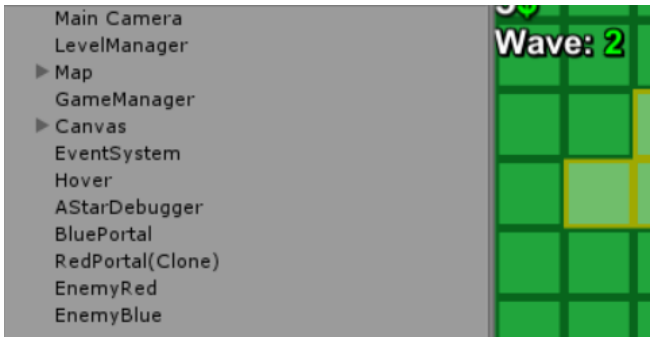
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ObjectPool : MonoBehaviour
{
    [SerializeField]
    private GameObject[] objectPrefabs;

    private List<GameObject> pooledObjects = new List<GameObject>();

    public GameObject GetObject(string type)
    {
        foreach(GameObject go in pooledObjects)
        {
            if (go.name == type && !go.activeInHierarchy)
            {
                go.SetActive(true);
                return go;
            }
        }
```

The screenshot above shows an array being created for the game objects called object prefabs, this will store all of the monsters that are no longer being used. Next a list is created for pooled objects which is used in the loop below. This loop is used to take each gameObject stored in the pooled object list and sets them active when they are needed in the game.

```
        for (int i = 0; i < objectPrefabs.Length; i++)
        {
            if (objectPrefabs[i].name == type)
            {
                GameObject newObject = Instantiate(objectPrefabs[i]);
                    pooledObjects.Add(newObject);
                newObject.name = type;
                return newObject;
            }
        }
        return null;
    }


    public void ReleaseObject(GameObject gameObject)
    {
        gameObject.SetActive(false);
    }
}
```

Above shows the rest of the object pool class, this image contains a loop for the object prefabs list and instantiates them in the game world, this also adds them to the pooled objects list for later use. The last part of the code is used when an object is removed from the game but not from the object pool.





These two images show waves 1 and 2 of a game. In wave 1, one red enemy was spawned. It was then added to the hierarchy. In wave 2, one red and one blue enemy were spawned. But in the hierarchy there is only 1 red enemy. This is because it was stored after being removed between the waves. This means the same red enemy was used for both waves, this is useful as on a larger scale when I tested the game without this function the game began to get laggy as it was trying to render too many sprites at the same time. This hotfix prevents that from happening.

Game Manager

The game manager class is one of the major constructs in my project, this maintains the in game experience and controls a majority of aspects. This class contains the variables for the wave number, the amount of lives a player has, the amount of money they have. It handles when the player loses the game and it links all the in game buttons on the GUI to the variables in the code.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class GameManager : Singleton<GameManager>
{
    public TowerBtn ClickedBtn { get; private set; }

    private int currency; // variable for money
    private int wave = 0;
    private int lives;
    private bool gameOver = false;
    [SerializeField]
    private Text livesTxt;

    [SerializeField]
    private Text currencyTxt;

    [SerializeField]
    private GameObject waveBtn;

    [SerializeField]
    private Text WaveText;

    private List<Monster> activeMonsters = new List<Monster>();

    public ObjectPool Pool { get; set; }
```

This first screenshot shows the libraries used by this class, as it is more complex it uses a considerable amount of commands so requires more libraries. The screenshot shows the variables for in game and the text variables for the buttons.

```
[SerializeField]
private GameObject gameOverMenu;

public bool WaveActive
{
    get { return activeMonsters.Count > 0; }
}

public int Currency
{
    get
    {
        return currency;
    }

    set
    {
        currency = value;
        this.currencyTxt.text = value.ToString() + "<color=lime>$</color>";
    }
}
```

The above image shows the links between in game money and the currency variable within the code, allowing for the game to be updated in real time. It also handles the boolean value for whether the wave is active, from here it returns the amount of enemies active.

```
public int Lives
{
    get
    {
        return lives;
    }

    set
    {
        this.lives = value;

        if (lives <= 0)
        {
            this.lives = 0;
            GameOver();
        }
        livesTxt.text = "HP: " + lives.ToString();
    }
}
```

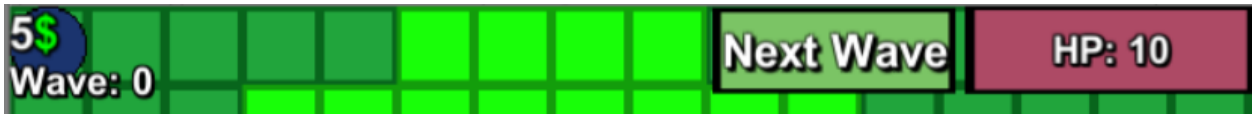
```
private void Awake()
{
    Pool = GetComponent<ObjectPool>();
}
```

This image shows how the Lives variable is linked to the in game text called HP and updates it when changes are made to the users Lives. It also defines the object pool and stores it as a variable called Pool.

```
void Start ()
{
    Lives = 10;
    Currency = 5;
}

// Update is called once per frame
void Update ()
{
        CancelTower();
}

public void PickTower(TowerBtn towerBtn) // new function called towerBtn which takes towerBtn into it
{
    if (Currency >= towerBtn.Price && !WaveActive)
    {
        this.ClickedBtn = towerBtn;
        Hover.Instance.Activate(towerBtn.Sprite);
    }
}
```

Next is the image above which shows how when the game is started Lives and Currency are assigned values, this then leads into a function for picking a tower. The PickTower function is applied to the tower button and ensures they can only select a tower between waves and if they have enough money. If they meet this criteria then the tower is attached to the cursor using the hover function.

The above image also shows how when the game starts the user will start with 10 HP and $5. This can be demonstrated with the test below.




```
public void BuyTower()
{
    if (Currency >= ClickedBtn.Price)
    {
        Currency -= ClickedBtn.Price;
        Hover.Instance.Deactivate();
        ClickedBtn = null;
    }
}

private void HandleEscape()
{
}

private void CancelTower()
{
    if (Input.GetKeyDown(KeyCode.Mouse1))
    {
        Hover.Instance.Deactivate();
    }
}
```

This image shows more functions within the game manager. Firstly is the BuyTower function which updates the Currency variable by subtracting the cost of the tower from it, then it resets ClickedBtn.

Next is the CancelTower function, this used to cancel placing a tower down when one has been selected. It does this by seeing if the user right clicks, if they do then it deactivates the hover.

```
public void StartWave()
{
    wave++;
    WaveText.text = string.Format("Wave: <color=lime>{0}</color>", wave);

    StartCoroutine(SpawnWave());

    waveBtn.SetActive(false);
}
```

The function above is used to start the wave, when the wave button is clicked it starts a coroutine called SpawnWave. This also disables the wave button while the game is active.

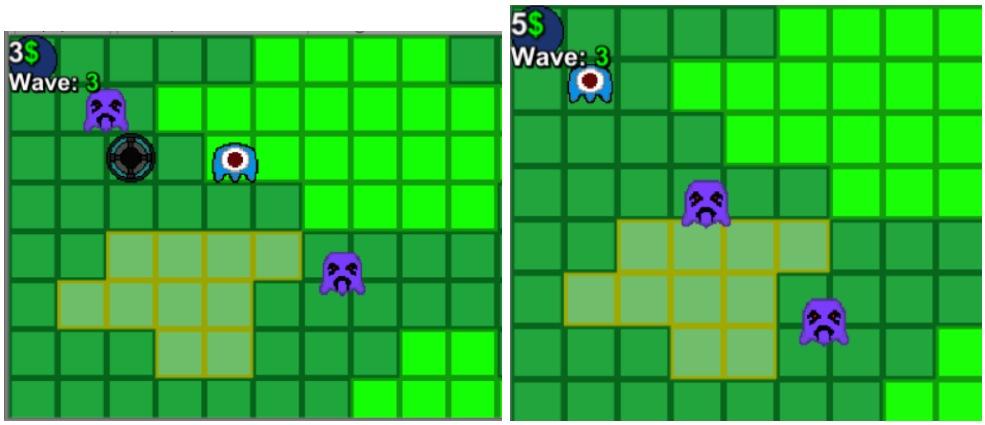
The images above show how the next wave button disappears when it has been pressed and while a wave is active.

```
private IEnumerator SpawnWave()
{
    LevelManager.Instance.GeneratePath();

    for (int i = 0; i < wave; i++)
    {
        int monsterIndex = Random.Range(0, 3);

        string type = string.Empty;

        switch (monsterIndex)
        {
            case 0:
                type = "EnemyBlue";
                break;
            case 1:
                type = "EnemyRed";
                break;
            case 2:
                type = "EnemyPurple";
                break;
        }

        Monster monster = Pool.GetObject(type).GetComponent<Monster>();

        monster.Spawn();

        activeMonsters.Add(monster);

        yield return new WaitForSeconds(2f);
    }
}
```

Spawn wave is a coroutine which increments the wave number by 1, it then uses the current wave number to spawn that amount of enemies. It spawns the enemies by using a random number from 0 to 3 (which doesn't include 3 as that is how unity works, encountered and error using 0 to 2). The random number then refers to an index of a list of monsters which it then adds to the object pool before being sent to the spawn function. Then as another fix I added a wait before repeating the monster spawn to prevent them spawning on top of each other which is an issue I encountered previously.

```
public void RemoveMonster(Monster monster)
{
    activeMonsters.Remove(monster);
    if (!WaveActive && !gameOver)
    {
        waveBtn.SetActive(true);
    }
}

public void GameOver()
{
    if (!gameOver)
    {
        gameOver = true;
        gameOverMenu.SetActive(true);
    }
}

public void Restart()
{
    Time.timeScale = 1; // for pausing
    SceneManager.LoadScene(SceneManager.GetActiveScene().name);
}

public void QuitGame()
{
    Application.Quit();
}
```

The last functions used in the game manager control the removal of monsters that have reached the end using the RemoveMonster function. Next is the GameOver function used when the player runs out of lives, this brings up the game over menu which has two options. The two options are restart and quit which are both handled in the last two functions of the game manager.



This image shows the amount of enemies being spawn is equal to the wave number, it also shows wave 3 on two occasions and shows that the enemies that are spawned are random.

A* Algorithm

The last classes for my program are the two I use for the A* algorithm. The first class is called Node and it is basically responsible for obtaining values for each node.

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Node
{

    public Point GridPosition { get; private set; }

    public TileScript TileRef { get; private set; }

    public Vector2 WorldPosition { get; set; }

    public Node Parent { get; private set; }

    public int G { get; set; }
    public int H { get; set; }
    public int F { get; set; }

    public Node(TileScript tileRef)
    {
        this.TileRef = tileRef;
        this.GridPosition = tileRef.GridPosition;
        this.WorldPosition = tileRef.WorldPosition;
    }
```

The first image shows the class just obtaining values for a tiles position in the grid, position in the game, parents of that tile and G,H and F values. These G,H and F values will be used to calculate a value for each tile in order to compare all adjacent tiles to the current node.

```
public void CalcValues(Node parent, Node goal, int gCost)
{
    this.Parent = parent;
    this.G = parent.G + gCost;
    this.H = ((Math.Abs (GridPosition.X - goal.GridPosition.X)) +  Math.Abs((goal.GridPosition.Y - GridPosition.Y))) * 10;
    this.F = G + H;
}
```

This section of code is how the Node function calculates a value for each Node, the F values of each node in the openlist of the A* are compared and the smallest value is selected.

Moving onto the AStar class next, this is where the majority of my projects complexity lies. This function identifies all nodes surrounding each current node, starting from the blue portal. It then compares each adjacent node and finds the shortest path to that node. The current node is then moved to the selected node and process repeats until the current node is equal to the final node.

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;

public static class AStar
{
    private static Dictionary<Point, Node> nodes;


    private static void CreateNodes()
    {
        nodes = new Dictionary<Point, Node>();

        foreach (TileScript tile in LevelManager.Instance.Tiles.Values)
        {
            nodes.Add(tile.GridPosition, new Node(tile));
        }
    }

    public static  Stack<Node> GetPath(Point start, Point goal)
    {
        if (nodes == null)
        {
            CreateNodes();
        }
```

This image shows the AStar function creating a dictionary of nodes which take two parameters; Point and Node. Then it moves on to a function called CreateNodes, this function is in charge of adding each node to the dictionary. The last section refers to when the GetPath function is called in the Level Manager, here it then runs the CreateNodes function if it does not already have a list of nodes.

```
HashSet<Node> openList = new HashSet<Node>();
Node currentNode = nodes[start];
openList.Add(currentNode); // adds start node to openlist

HashSet<Node> closedList = new HashSet<Node>();

Stack<Node> finalPath = new Stack<Node>(); // Actually using a stack in CPU memory for backtracking in the final A*

while (openList.Count > 0)
{
    // for loops to search through all adjacent nodes to current node
    for (int x = -1; x <= 1; x++)
    {
        for (int y = -1; y <= 1; y++)
        {
            Point neighbourPos = new Point(currentNode.GridPosition.X - x, currentNode.GridPosition.Y - y);

            if (LevelManager.Instance.InBounds(neighbourPos) && LevelManager.Instance.Tiles[neighbourPos].WalkAble && neighbourPos != currentNode.GridPosition)
            {
                int gCost = 0;

                if (Math.Abs(x - y) == 1) // Decides g value for node whether it is vertical or diagonal
                {
                    gCost = 10;
                }
                else
                {
                    if (!ConnectedDiagonally(currentNode, nodes[neighbourPos]))
                    {
                        continue;
                    }
                    gCost = 14;
                }
```

After it has done this the function then sets the current node equal to the start node, this then leads into the closed list being created for the final path which created on the line below as a stack. It is created as a stack as it is the easiest method I could find for accessing data stored, as it is a first in first out system it is ideal for this algorithm.

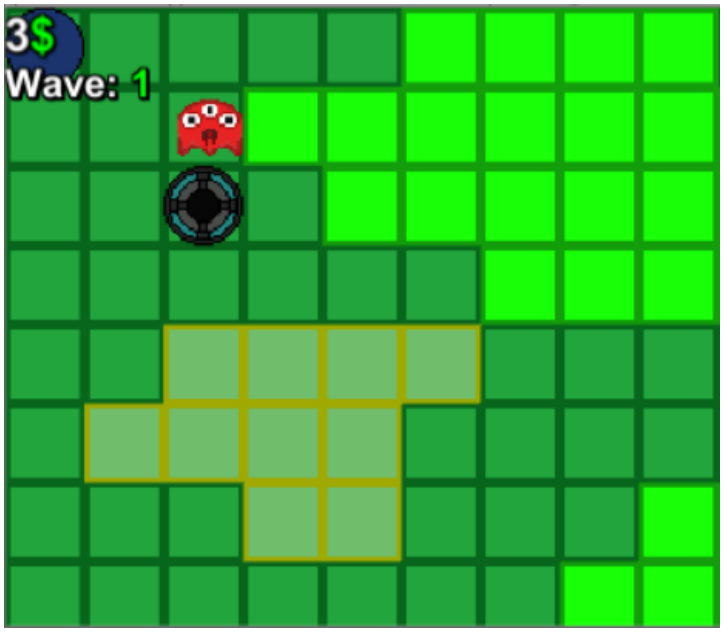
Next it analyses the openlist of nodes which are all adjacent to the current node, from here it then assigns each neighbouring node a gCost depending on whether the node is diagonal or adjacent to the current node.

```
                Node neighbour = nodes[neighbourPos];

                if (openList.Contains(neighbour))
                {
                    if (currentNode.G + gCost < neighbour.G)
                    {
                        neighbour.CalcValues(currentNode, nodes[goal], gCost);
                    }
                }

                else if (!closedList.Contains(neighbour))
                {
                    openList.Add(neighbour);
                    neighbour.CalcValues(currentNode, nodes[goal], gCost);
                }
            }
        }
    }

    openList.Remove(currentNode);
    closedList.Add(currentNode);

    if (openList.Count > 0)
    {
        currentNode = openList.OrderBy(n => n.F).First(); // orders each nodes n by f value, take first value on list
    }

    if (currentNode == nodes[goal])
    {
        while (currentNode.GridPosition != start)
        {
            finalPath.Push(currentNode);
            currentNode = currentNode.Parent;
        }
        break;
    }
}
return finalPath;
//GameObject.Find("AStarDebugger").GetComponent<AStarDebugger>().DebugPath(openList);
}
```

The final part of this function is shown above, this then compares the current node to the neighbour nodes' gCost value. The smallest gCost node in the neighbour list is then replaced as the current node, the previous current node is then added directly to the closed list for the final path and is removed from the open list. The function then orders the openlist and takes the first value, if this value is equal to the goal node then it pushes the current node into the final path. Finally this returns the final path.

```
// checks if A* is allowed to move diagonally
private static bool ConnectedDiagonally(Node currentNode, Node neighbour)
{
    Point direction = neighbour.GridPosition - currentNode.GridPosition;
    Point first = new Point(currentNode.GridPosition.X + direction.X, currentNode.GridPosition.Y + direction.Y);
    Point second =  new Point(currentNode.GridPosition.X, currentNode.GridPosition.Y + direction.Y);

    if (LevelManager.Instance.InBounds(first) && !LevelManager.Instance.Tiles[first].WalkAble)
    {
        return false;
    }
    if (LevelManager.Instance.InBounds(second) && !LevelManager.Instance.Tiles[second].WalkAble)
    {
        return false;
    }
    return true;
}
```

This above function is used as I encountered some errors when the current node was on the edge of the map, as there was no nodes on some sides of it as it was at the border I needed to add some validation for whether it was inbounds. This function also works to ensure a node can move diagonally across a tile as if there is an unwalkable tile near it then it cannot move diagonally.



This image shows an enemy moving around a tower as it cannot path diagonally across it.

Camera Movement

Test No.	What is tested?	How is it tested?	Why it is tested?	Test Data			Expected Result	Actual Result
				Normal	Abnormal	Extreme		
1	WASD controls for camera movement	Visual monitoring of the game when these keys are pressed	Cannot navigate game world if not functional	W	F	/	Camera will pan across the game world	Camera controls work as expected
1.1	Limits of camera against map border	moving camera to edge of map	Camera should be locked only to game	Moving camera within game world	Moving camera near edges of game world	Moving camera only against edges and corners	Camera should stay locked to map limits	Potential Issues on different resolutions but works as expected

Tower Purchasing and Currency

Test No.	What is tested?	How is it tested?	Why it is tested?	Test Data			Expected Result	Actual Result
				Normal	Abnormal	Extreme		
2	If tower button is clicked, tower attaches to mouse	clicking a tower button	To show the user which tower they have clicked	N/A	N/A	N/A	Tower will attach an icon to the mouse	Works as expected
2.1	if right clicking when a tower is selected cancels that tower	Selecting a tower then right clicking	Gives user option to change their mind	N/A	N/A	N/A	Tower icon removed from cursor	Works as expected
2.2	If tower is placed with left click currency is changed	Placing a tower down with left click	To check currency is updated in real time	N/A	N/A	N/A	Currency decreases by tower cost	Works as expected
2.3	Towers cannot be placed on top of each other	Attempt to place two towers on same tile	To ensure towers cannot stack	N/A	N/A	N/A	Tile will highlight red and will not place	Works as expected
2.4	Towers can only be placed when they can afford the tower	By trying to buy a tower that costs more than currency	To prevent users from getting towers for free	N/A	N/A	N/A	Will not be able to purchase tower	Works as expected

A* Navigation

Test No.	What is tested?	How is it tested?	Why it is tested?	Test Data			Expected Result	Actual Result
				Normal	Abnormal	Extreme		
3	Start node and end node are identified	Using a debugger that highlights start and end	to check the path starts and ends at correct points	N/A	N/A	N/A	They will both be highlighted green and red respectively	Works as expected
3.1	Pathway is shortest path available	Recreate the map in a known A* and compare paths	To make sure the complexity of the program is accurate	N/A	N/A	N/A	Shortest path will be calculated	Shortest path found and used
3.2	Objects such as towers are not included in path	Debugger shows pathway not including towers	To prevent monsters colliding through towers	N/A	N/A	N/A	Towers will be avoided by enemies	Towers are avoided by enemies on path
3.3	Cannot create diagonal connections across blocked tiles	Debugger shows path avoids diagonal connections near unwalkable tiles	To prevent monsters colliding through towers	N/A	N/A	N/A	Towers will not be cut across by enemies	Towers cannot be crossed diagonally
3.4	Path is updated between waves	Path is observed before and after changes are made between waves	To make sure monsters path around any changes made such as new towers	N/A	N/A	N/A	Path is recalculated to stop monsters crossing over new towers	Path is reset between waves to account for changes to map
3.5	Path is applied equally to all enemies that spawn	Check to see if all enemies follow same path	To see if all monsters follow the same path	N/A	N/A	N/A	Enemies will all follow the same path	Works as expected

Monster Spawning and Pathing

Test No.	What is tested?	How is it tested?	Why it is tested?	Test Data			Expected Result	Actual Result
				Normal	Abnormal	Extreme		
4	Monsters spawn randomly	Check if there is use of random integers in code for spawning	To add some variety to the waves	N/A	N/A	N/A	Monsters spawn randomly	Works as expected
4.1	Monsters spawn at blue portal	See if monsters appear from blue portal	Because enemies should only spawn from the start point	N/A	N/A	N/A	Monsters spawn at blue portal	Works as expected
4.2	Monsters despawn at red portal	See if monsters disappear at red portal	To check that monsters are removed at the end	N/A	N/A	N/A	Monsters disappear at red portal	Works as expected
4.3	Monsters despawn when destroyed	Observe whether monsters disappear when attacked by tower	To make sure monsters do not stay after being destroyed	N/A	N/A	N/A	Monsters disappear when destroyed by tower	Monsters are never destroyed due to lack of tower functionality
4.4	Monsters are reused to prevent lag	Use a collider to find game objects and observe if it increases when no new monsters are observed	To reduce processing power that could be spent on other aspects of the game.	N/A	N/A	N/A	Monsters will be reused	Works as expected
4.5	Monsters are not spawned on top of each other	Visually observe when monsters spawn	To stop monsters from overlapping as it reduces the quality of the game	N/A	N/A	N/A	Monsters wait between spawns to prevent overlap	Works as expected, time delay between each enemy spawn

Tower Shooting

Test No.	What is tested?	How is it tested?	Why it is tested?	Test Data			Expected Result	Actual Result
				Normal	Abnormal	Extreme		
5	Tower shoots projectile at monster	See if projectile is fired at monster in adjacent tile	To make sure towers attack enemies	N/A	N/A	N/A	Tower shoots at monsters	Towers do not fire projectiles
5.1	Tower only fires projectile in certain radius	Place towers at different distances and see if all attack	To see whether towers range works correctly	N/A	N/A	N/A	Towers only shoot in their range	Radius feature not implemented
5.2	Tower fires set amount of projectiles per minute	Count how many times tower fires projectile	To balance the game and to make sure it isn't too easy	N/A	N/A	N/A	Towers only fire 20 times in a minute maximum	Towers do not fire projectiles
5.3	Projectiles are removed after fired	Observe unity hierarchy and see if they are removed	To prevent lag as too many entities can cause performance issues	N/A	N/A	N/A	Projectiles are removed after they are fired	Projectiles are never created so they are not removed

New Waves and Between Waves

Test No.	What is tested?	How is it tested?	Why it is tested?	Test Data			Expected Result	Actual Result
				Normal	Abnormal	Extreme		
6	Next Wave button works	See if enemies are spawned when pressed	To make sure another wave can start	N/A	N/A	N/A	New wave will spawn when pressed	Works as expected
6.1	Towers cannot be placed between waves	Try to place a tower between waves	To prevent issues with A*	N/A	N/A	N/A	Towers will not be able to be placed	Towers can only be placed between rounds

6.2	Enemies spawned is equal to wave number	Observe amount of enemies per wave and compare	Because it is a game feature	N/A	N/A	N/A	Enemies spawned will be equal to wave number	Works as expected

Health Points

Test No.	What is tested?	How is it tested?	Why it is tested?	Test Data			Expected Result	Actual Result
				Normal	Abnormal	Extreme		
7	Health points decrease when enemies reach end portal	Observe if health points decrease when enemies reach end point	Because it is a game feature and the user must attempt to prevent this	N/A	N/A	N/A	Health points will decrease when enemy reaches end portal	Health points decrease when enemy reaches end portal
7.1	Always start with 10 health points	Check at the start of the game	To make sure all users always have a fair start	N/A	N/A	N/A	User will always start with 10 HP	User starts with 10HP
7.2	Game Over menu appears when health points reaches 0	See if menu appears when HP reaches 0	To allow the user to restart or quit the game	N/A	N/A	N/A	Game over menu will appear at HP=0	Menu appears when HP reaches 0

Post Development Testing

Module	Rating (1 - Lowest 5-Highest) Reason Optional	Feedback
Camera Movement	This was reviewed by 3 different stakeholders in my class who rated the camera movement 4,5,5 respectively. My peers have said this is because it functions as described, it moves smoothly across the screen, it doesn't leave the game boundaries except there is no zooming feature.	Positive: Smooth camera movement and perfectly locked to game world
		Negative: Moves relatively slowly
		Improvements: A zooming feature for the camera
Tower Purchasing and Currency	This was tested multiple times by myself throughout the development as well as my teacher and students from my class. The tower purchasing was easy to do and easy to understand. It was rated 5 out of 5 by all testers.	Positive: The towers were purchased and placed easily and the currency updated straight away. Currency also wasn't changed when a tower was cancelled.
		Negative: No information about towers when you hover over them on the icons on the right of the screen.
		Improvements: Add information about towers when user hovers over tower icons.
A* Navigation	The A* is the main complexity of	Positive: Perfectly navigates

	my project and I worked very hard on ensuring that it would work smoothly without issues. I was however limited as I couldn't figure out how to apply the unwalkable variable to tiles which limited the ability to create a specific path I wanted the monsters to follow. Overall rating was 4 from peers.	from start to finish and avoids towers that are in the way of the shortest path.
		Negative: A* was not applied to certain tiles meaning it could only avoid towers not tiles.
		Improvements: Apply unwalkable variable to tiles to limit the area the monsters can path across.
Monster Spawning	Monsters spawned smoothly from blue portal and scaled up to normal size. Enemies spawned was equal to wave number making the game increasingly harder. Rating for this module was a 4.	Positive: Enemies spawned without error and despawned smoothly also.
		Negative: Enemy monsters didn't look fantastic graphically.
		Improvements: Spend more time on enemy design to make them look better.
Tower Shooting	Due to time limitations I was unable to program fully functional projectiles that would fire from the tower at enemies. This has held the project back but due to other subjects my time had to be spread out so I couldn't apply all of the features I aspired to have in this project. Rating 1.	Positive: N/A
		Negative: Function not implemented
		Improvements: Implement function into towers.
New Waves / Between Waves	Wave system worked successfully and the user could easily press the next wave button. When the button was pressed tower placing was suspended as intended which was also good. Rating 5.	Positive: Next wave button worked perfectly, expected features were successfully suspended when wave was in progress.
		Negative: Maybe add a way to access the options menu between waves.

		Improvements: Add the ability to access the options menu between waves in case the use wants to restart or quit.
Health Points	The health points system worked successfully and decreased by 1 for every enemy that reached the end red portal. As there was no way to prevent the enemies from reaching here as towers could not attack the system was limited. Rating 3.	Positive: The health points system worked successfully for when enemies reached the end. The game over menu also successfully appeared when HP = 0.
		Negative: There was no way to prevent the enemies from reaching the end point without blocked the pathway with towers.
		Improvements: Finish the tower shooting functions in order to take full advantage of the HP system.

Main issues discovered after testing took place revolve around the lack of the towers offensive capabilities. This significantly limits the game as functions more as a A* path blocking simulator. While this limitation inhibits a lot of the gameplay it does however show that while balancing multiple A levels time must be spread equally, so while it is unfortunate that this has happened, the features required to finish the game will be added in any spare time I have after examinations are over as the whole project has been a fantastic experience.

Section 4

Success Criteria

Primary Goal	Criteria for goal	Criteria met/not met	Primary Goal met/not met
Allowing the User to access a settings menu from the game	Settings Menu can be accessed in the game	Met, when the game is over a menu appears that gives the user opportunity to quit the application or restart	Primary goal met partially as throughout the game the user does have the opportunity to access a menu. This menu is presented at the end of the game and while limited in functionality it still provides the user some options on whether they want to restart or quit. My stakeholders George Bond and Robert Allan were the main evaluators for this module and they agree that the primary goal is met, this is due to the functionality provided by the menu upon losing the game.
	User can change sound settings	Not met due to lack of time allowed for project due to other subjects commitments, this led to a lack of music and sound for the game so no need for a menu for them.	
	User can change difficulty settings	Not met as the game's difficulty increases progressively, this means there is no need for the option as the game starts slow and increases exponentially.	

GUI of the application is easy to understand and an explanation is available to the user in game	User can access in game explanation	Not met, GUI was designed fairly simply and self explanatory so an in game explanation didn't seem necessary when other modules took priority.	Primary goal met as the GUI is easily navigable and self explanatory from how it is structured. This has been evaluated and agreed with by my stakeholder James Foggin.
	User can comfortably use and navigate the GUI	Met, GUI is relatively simple and is quickly picked up by my peers who have used it.	
	GUI will be self explanatory and will not require explanation	Met, this clause makes the first criteria relatively redundant but the GUI is simple enough to understand from only a brief try.	
Application provides fun and challenging experience and is balanced	Game is balanced and fair making it completable	Not met as the game currently has no way of being completed	Not met unfortunately due to the lack of the towers shooting module. However with that module implemented the game would be playable and enjoyable. This was decided to have not met the criteria by my stakeholder Robert Allan.
	Application is challenging and self learning	Met as the game is self learning with the A* algorithm, its difficult also increases as it is played for longer making it challenging	
	Application has depth and provides an enjoyable experience	Not met because while it may have depth it still lacks a few essential features to make the game playable.	

Users score will be stored in a data file which can be sorted and return the top 3 scores	Users score is stored after completing the game	Not met	This section is not meeting the criteria due to the lack of essential modules in order for a score to be calculated. This section was evaluated by myself as other stakeholders are not required as the level of success is quite apparent.
	User can access and view other scores in the data file	Not met	
	User can view the highscores people have achieved	Not met	
User can access a variety of towers within the game	User can purchase and place towers on the map	Met as towers can be purchased and placed successfully	Mostly met as the majority of the criteria were met very well, the module still suffers like most of the project from the common theme of the shooting module, which has held back the project quite a lot. As the level of success was hard to determine in this module I evaluated it with stakeholders George Bond and James Foggin, together we discussed the extent at which this was successful and concluded this module as described above.
	User can place multiple different towers on the map	Met as if the player can afford multiple towers then they can be placed on the map just not on the same tile.	
	User has access to multiple towers with different features	Not met, while the game has complexity due to time constraints and the lack of the shooting module I was unable to get chance to customise individual towers.	

To improve upon the criteria of my game it would have to rely less on content of the game and more the complexity of the programming, this could have been accomplished by focusing on other modules such as a login screen, a pause menu, a database of users, a way to register a new user and other varieties of features that would demonstrate my technical ability.

Stakeholder Review

Stakeholder	Review of project (Positives and Negatives)	Points to improve upon	Did the program achieve its goal
George Bond	**Positives**: The project has significant complexity due to the A* algorithm, this along with multiple small features that increase the quality of the product, such as the validation for tile placing and the optimisation using the object pool **Negatives**: With the lack of the modules that would allow towers to have more functionality the game is limited, while it is complex and learning it lacks some content.	By working on the tower modules that could increase the content within the game and the functions that would benefit from those changes the program could be taken to another level.	The goal of the program was to create a tower defence game that user an A* pathing algorithm, the player had to prevent the enemies from reaching the end point in order to play well. While the towers may lack in the ability to shoot they can be used to block the end point which does still follow the criteria of the game.
Daniel Rigby	**Positives**: The program that I have experienced provided an exciting experience into what could be a fun strategy game, while it lacked some features the features that were available all functioned great. **Negatives**: The game was missing some content that would have made it a fair bit better, the missing features however are relatively complex and it is understandable they were missing	Better time management could have potentially allowed for more content to be added to the game, while spending time on a project is one thing, how you spend that time effectively is just as important.	The program was very aspirational with hopes of lots of complex features, while this may have been a little too much work to pull out the project still came out quite well. The program consists of many fully functioning and optimised code which is impressive I would say the program was not far off completing its' goal it could have just used a few.more features to get it there.

Jude Taylor	**Positives**: The GUI of the finished product while a little crude around the edges (which is to be expected as not everyone is an artist) came out very good. The buttons and menus were all scaled appropriately and all scaled properly on different resolutions. The GUI is easy to understand and simple to use. **Negatives**: The GUI while it functions quite well could potentially use some cleaning up, maybe using some better textures on buttons and consider adding some sound effects for when they are pressed.	The game while unfinished functions exceptionally at the features it has. I think to take the game further apart from the obvious points of introducing new modules that if the game had some music and sound effects it would be much more enjoyable.	I would say the program achieved its goal of demonstrating a complex A* pathing algorithm and while it lacks certain features the game has still got many features built into it that give the game a nice feeling. The highlighting of the tiles when a tower hovering is a fantastic feature that you don't always see in similar games. I think with more time this could have been an amazing piece of software.

Summary of Stakeholder Review

The stakeholders for my project all agreed that it was lacking some content, and while this is true some of them still believed the game had successfully become the intended product from the start. Many games these days are released early with less features that intended and are updated as they are developed, constantly changing and getting more complex. This means that the game can grow with the community and the feedback from the players can provide valuable insight into new potential features, issues that must be seen to and things they don't quite like. The industry revolves around this these days and most modern games are given regular updates, patches and bonus content that is added to them. With that in mind my stakeholders have highlighted some issues with the project but have also discussed some of the great features it has.

Strengths and Limitations

My stakeholders have concluded that the games quality currently revolves mostly around the complexity of the algorithm involved as well as the GUI, tower purchasing and tower placing on the grid. They have also expressed lots of positive feedback with the use of my object pool to "recycle" gameobjects as this optimises my code quite

nicely. From the feedback I received they seem impressed with the use of an external text file for map generation, this is because the way I am generating my map means that it can easily be changed, modified or extended. This allows for loads of opportunity for multiple levels as I could use the same method for choosing random monsters for choosing a random map at the start of the game. This way I could procedurally generate maps with little difficulty.

When creating this program I was mostly limited by my available time and lack of knowledge using this new language, I struggled to implement certain features due to my inability to create the code for it and struggled further when it came to implementing large new modules that would add a lot to the game. While creating this project I have had the workload of 3 A levels to contest with while I am proud with my finished product, I know with more time I could have implemented many more game features to take it to a whole new level. This has held back the quality of my project to an extent and I think with better time management I would have been able to make it better, but from what I have made I have learned valuable skills about modular programming and research into new programming constructs. At the end of my project without being able to add in too many large features I did manage to add in a few small ones. These small features include cancelling tower placements with right click in case the user changes their mind, a restart and quit feature that is offered at the end of the game, and finally instead of set waves of enemies I changed them to random spawns to add some variety. I managed to do this by coming back to the code every week with a new idea and used some debuggers I created on copies of my original code to test whether I could implement these new ideas. After multiple failed attempts I found ways to add these features in and it has made me a more confident programmer since.

Maintenance

In order for the program to continue to work I will be maintaining it will new content that will be added in updates whenever I get time to work on the project more. This means after my exams or potentially an hour here and there I will work on some of the modules of my program that would step it up and will hopefully be able to add them in within the following months. While this project is very rewarding my current priority is my other subjects so updates to my program may be inconsistent.

Maintaining a program over a long period of time will come with the issues of memory, without proper comments and notes throughout the code I may find myself lost in complex structures such as my A*. This will be challenging as I will need to keep my

mind fresh with my code to ensure I don't fall behind and lose the opportunity to finish and improve on my project.

Further Developments and Improvements

When coming back to this program I have a long list of features that I would like to add into my program to significantly improve the quality of it. This mostly includes adding more depth to the towers in terms of content. I will want to work on making the different towers have different abilities which will be used against the enemies. This will likely include things such as splash damage that will affect multiple enemies, burn damage that will cause them to take damage over time, slowing effects that inhibit and impact enemy movement across the game world.

After working on the complexity for the towers I would like to work on some basic sounds and music for the game, be it a simple beat and some plain sound effects it would still make the game feel much more realistic. This will definitely be a challenge as I have little experience with that type of work but it will be fun to figure out.

I think the last major improvements I would like to add to my game is multiple maps, be them procedurally generated using random numbers and a simple algorithm or preset maps that it will randomly select. These maps could also have map specific enemies such as zombie and undead monster themes for desert maps and night time maps to even aliens for a space themed map where tiles represent planets.

I have a lot of ideas for this project but right now my priorities are elsewhere, I am aware my project lacks some depth in game content and graphically and these are going to be some important areas I would like to focus on in the future.

Bibliography:

Current Systems Research:
https://www.commonsensemedia.org/app-reviews/plants-vs-zombies

Statistics About Gamers of Current System:
https://forum.supercell.com/showthread.php/277511-How-Old-Are-Most-Clash-Of-Clan-Players