Contents

Tower Defence Game Analysis

Ghusharib Chohan 605

The project:

For my project, I will be creating a tower defence style game. The game will revolve around the storyline of Hansel and Gretel, where they have both escaped the witch but are still stuck in the house, being attacked by the wildlife around them. They, therefore, need to use the candy surrounding the house and the weapons in the house itself to defend themselves from attacks and progress to leave the forest back to their father.

The game will have multiple levels, with each level being harder than the last and will give the user a set number of lives. Once they have lost all their lives, they would have to restart the game. The weapons would need to be purchased using in-game money and this money would be earned by destroying enemies. The weapons would be placed around the house and on the porch etc. and once the user hits play, the rest will be simulated. The attackers will slowly advance and the weapons would fire, reducing their health. If the attacker's health reaches zero, they die but if they reach the home, the house's health is reduced. This will be repeated across the levels.

This increasing difficulty and health level means that the game can be simulated to see the effects of the user placing various weapons in certain places. In addition, the interactive part of placing weapons means that it too can be programmed as a click and drag on a screen.

This game is programmable, as it would require a click and drag function to drop weapons in specific places, and a count of all the players' health and the attackers' health. In addition, the game will require a loop to simulate the attack and how well the weapons defend Hansel and Gretel. This will also require programmed animations and mathematical calculations, as well as scheduling to progress the game. In addition, file management will be required to save the game and save high scores/keep track of a user's progress and score.

I will also use a RAD approach to my project as it is the most suitable for it. By creating multiple prototypes and continuously adding functionality, I will be able to gradually improve my game as well as add the complex pieces of code such as scheduling. In addition, this will allow me to break down a fairly complex program into smaller parts which I can program individually and so a Computational Approach is the best way to my project.

My stakeholders

The users of my game will target many types of people. Those aged 7 and over who enjoy a strategy game will be constantly attempting levels, while commuters aged above 18 may also enjoy playing the game on their journey to/from work. These users are suitable because the game aims to be a quick and short game with levels of increasing difficulty. It also is a fairly casual game that doesn't require too much thinking but can keep a user hooked for a fair amount of time, and so is perfect for all ages and all capability.

This project is aimed to give the end user satisfaction of completing the game and enjoyment when they are relatively bored. As it shouldn't take up too much of the user's time, it also means that it is a very flexible game and users won't feel as though they are being forced to play for long hours and can choose how long they wish to play for.

As agreed by those listed, I will be helped by these people in terms of testing etc. as they are a good target market. They are all aged around 17-19 and are regular commuters. This means that they will be willing to play the game for short bursts when they are bored, making them the perfect testing focus group. My group is made of 3 Computer Science Students and 3 General Gamers:

Michael Kuc – Being a fellow Computer Science student, he is likely to have a keen idea for the kinds of bugs that could come about when programming such a complex game. This means that he will notice things that others in my focus group may not necessarily pick up.
Nathan Wang – Has a keen eye for the design of menus and screens etc. If he doesn't feel that something is usable, he will make sure that I know about it and I improve my look and feel of the game (especially in terms of menus)
Oliver Wales – Has a tendency to look out for common bugs and so will notice if there is something that isn't working quite right in the game (even if it is small)

Praveen Murugathas – Generally someone who enjoys playing games and so will be focussed largely on the gameplay and graphics in terms of how well they work together. This should help give me the feedback I need in terms of the mechanics of the game. Praveen is the main user as will give the right feedback in enough detail compared to the other two gamers who will give general feedback.
Vinayak Shastri – Another general gamer who will be focused on the entertainment side of the game
Miron Abhayasinghe – Another general gamer who will give feedback on whether the game is interesting to play or just another regular tower defence

These stakeholders will be able to interact with me through interviews and my blog on www.ghusharibcomputinggame.wordpress.com .

Researching the problem

Firstly, I required some background knowledge into the history of tower defence games and what exactly they are. I did some research online to find such ideas, as there are many forums dedicated to tower defence and other strategy games. In addition, there are many people who have good ideas and post them online in blogs which I thought I could benefit from. My research can be summarised by the points below:

[Source https://artofgame.wordpress.com/2009/04/14/lets-talk-about-tower-defense/, Date Accessed [12/09/2017]]

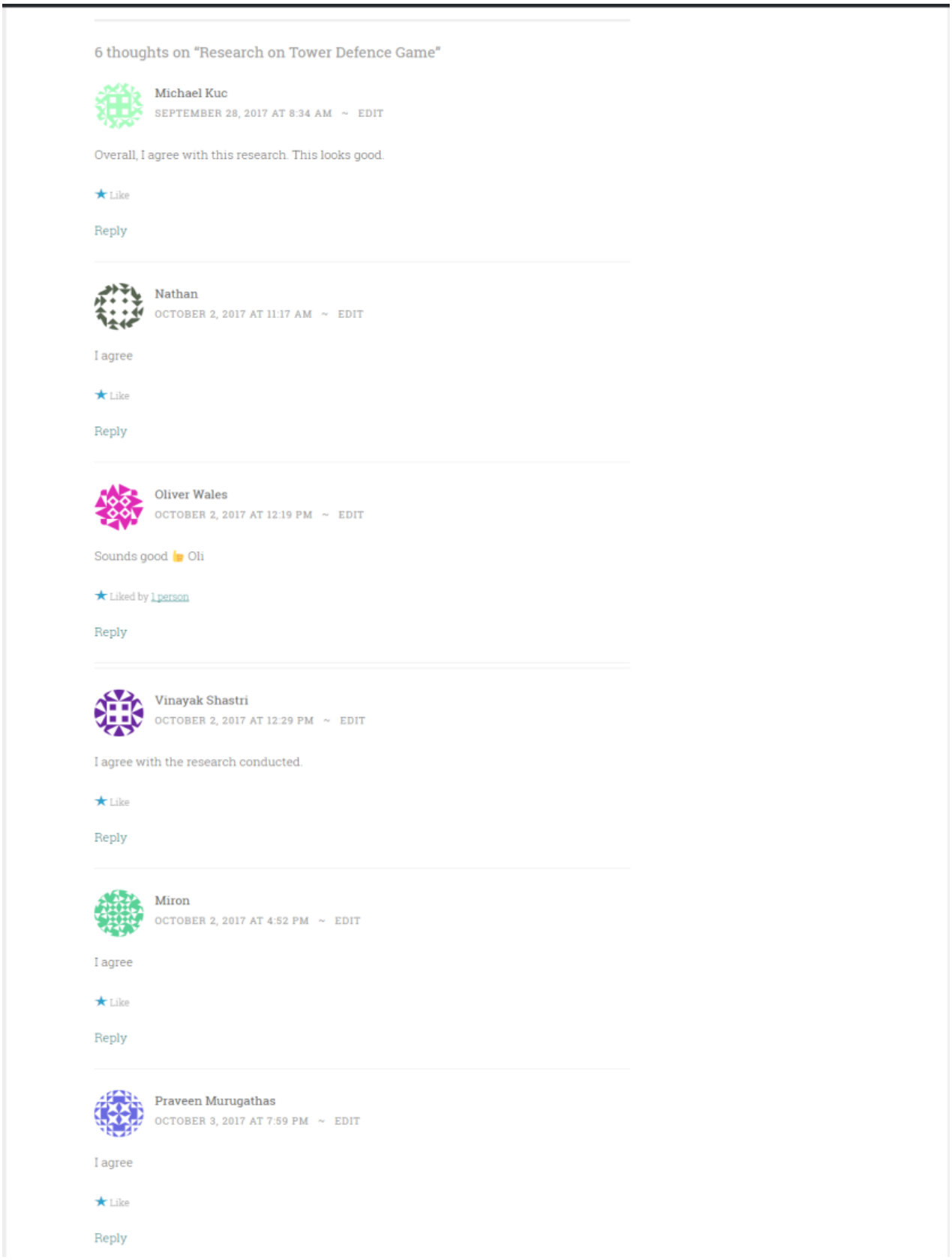
- Tower defence games tend to be endless until failure. This means that I would require my game to have increasing difficulty by itself, and not have a set number of levels.
- It is a strategy game, and strategy needs to be a clear way to win the game when designing it, and so this will require a lot of planning.
- Weapon option is a huge consideration. Having too many can ruin the game, but having too few can make it impossible. Each weapon must be different.
- Surprisingly, few TD games have enemies assaulting the towers, so this could be a key point to add to my own game
- Modifications such as magnets pulling enemies in a certain direction or treadmills aren't a thing, and these are key things that if added could prove to be a major improvement to a TD game.
- Speeding up the animation (or even slowing it down) is a key feature no tower defence game would be fun without.

[https://www.gamasutra.com/blogs/LarsDoucet/20140923/226261/Optimizing_Tower_Defense_for_FOCUS_and_THINKING__Defenders_Quest.php
Date Accessed [12/09/2017]]

The game may benefit from having a tower offence side of things as well. This is where the user itself is involved in attacking a base strategically rather than only defending the base.

A better way to research this problem is to go for interviews – preferably as individuals. This is because tower defence games are not a new idea, and have been around for many years. They are extremely popular but are beginning to see a downwards trend in activity due to its slightly tedious nature. Therefore, by interviewing people, I should be able to engage with the audience and ask for improvements. This requires question production and arranging interviews.

My stakeholders responding to the research I did:



6 thoughts on "Research on Tower Defence Game"

Michael Kuc
SEPTEMBER 28, 2017 AT 8:34 AM ~ EDIT

Overall, I agree with this research. This looks good.

★ Like

Reply

Nathan
OCTOBER 2, 2017 AT 11:17 AM ~ EDIT

I agree

★ Like

Reply

Oliver Wales
OCTOBER 2, 2017 AT 12:19 PM ~ EDIT

Sounds good 👍 Oli

★ Liked by 1 person

Reply

Vinayak Shastri
OCTOBER 2, 2017 AT 12:29 PM ~ EDIT

I agree with the research conducted.

★ Like

Reply

Miron
OCTOBER 2, 2017 AT 4:52 PM ~ EDIT

I agree

★ Like

Reply

Praveen Murugathas
OCTOBER 3, 2017 AT 7:59 PM ~ EDIT

I agree

★ Like

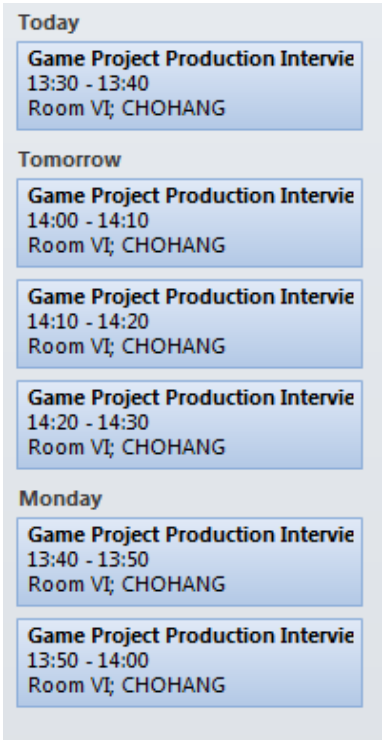
Reply

Questions to ask (Interview 1):

1) **How much time do you roughly aim to spend on a game/mission?**
   This first question is key to understanding what kind of game I need to make. Having missions would give me the benefit of giving my users the ability to play in short bursts, whereas having a long mission might put users off as they don't like to play a game for that long.
2) **Should the storyline be fully integrated into every part of my game?**
   These questions should help me understand whether or not the storyline is the best part of tower defence or if the gameplay is more important. I personally tend to skip storylines, but having a storyline running in the back may help integrate the users more. It is up to the user's to decide whether they would like a storyline or not, and therefore is an important question in the interview.
3) **Should a mission last longer than a minute, or should there be speed options to help with this.**
   Tower defence games were a big craze a few years back but aren't as popular now. This is due to their repetitive nature and purely because all tower defence games are the same. However, by understanding the user's reasons for not liking tower defence (if they don't), then I can adapt my game to overcome their worries.
4) **In terms of graphics, how important would you rate them to be in order to help make the game more addictive?**
   The game design is something which is complex to overcome. Some users prefer a professional looking menu and design, others prefer a basic easy to use design. This should help me get a flavour of what the average user likes.
5) **What kind of features are missing from modern day tower defence?**
   This is a very general and open question as my earlier research seemed to show that regular tower defence players have great ideas as to how to improve the game. This should help me understand what they want in a game.
6) **Should the power up be one time use only or multiple uses?**
   Power ups are liked a lot in other types of games, so by suggesting for them to come into tower defence as well, perhaps I will be creating a different type of tower defence. The number of uses helps understand how to balance the game in this way.
7) **Should the user be allowed to modify their defence during the simulation?**
   This is just a general question to ask users whether or not they are happy with the fact that a lot of tower defence games require you to just watch the defence be torn apart during the simulation rather than add to your defence.

Signing off the Interview Session:

Interviewee	Date/Time	Place
Michael Kuc	15/09/2017 @ 14:20	Face to Face
Nathan Wang	15/09/2017 @ 14:00	Face to Face
Praveen Murugathas	14/09/2017 @ 13:30	Face to Face
Oliver Wales	15/09/2017 @ 14:10	Face to Face
Vinayak Shastri	18/09/2017 @ 13:30	Face to Face
Miron Abhayasinghe	18/09/2017 @ 13:40	Face to Face



I attended and answered all the questions in my interview session 1 on the date stated:

_____

_____

_____

_____

_____

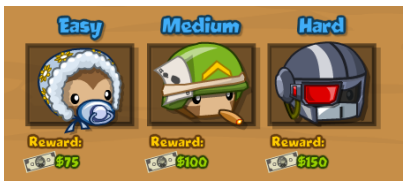
_____

From the answers I gathered from interviewing (See Appendix A), these were the key points which I decided to take forward as part of my research into a solution:

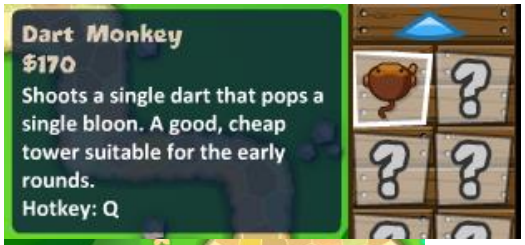
- Users tend to play tower defence for around 30 minutes up to 45 minutes during one sitting meaning that it should be possible to avoid having levels and instead, a never ending game
- Most users decided that storylines were an important feature as they helped explain the game, but in a tower defence game, an intro alone will do
- Tower defence games can become repetitive, and this is especially a problem when there is one clear strategy that can power you through all the missions
- The design should be simple and the menus should match with the graphics
- Common Features requested – Building Times, Emergency feature when the game is getting tough such as Spikes from Bloons
- Objectives, More than One Route, Enemies Fight Back
- Powerups would be nice, especially the Spikes feature from Bloons and other emergency powerups, but these should be limited to enable a balance to be reached. This should be done via reset times etc.
- Some people thought that yes, you should be allowed to add to your defence while the simulation is running whereas others disagreed. However, it is probably important to have an additional defence to enable the powerup features. Once again, this should not ruin the balance of the game.

I looked at other tower defence games to get ideas from them too, as my interviews referred me to games users liked such as Bloons. This gave me an idea of the look and feel of the game I should be aiming for in my own game:

Bloons TD5 offers multiple tracks to play on from the start



It also allows you to choose a difficulty level with different in game money rewards

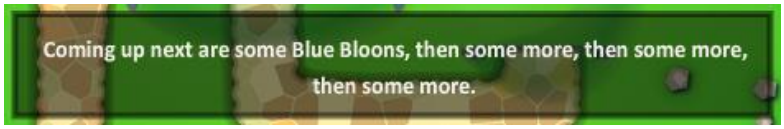


A complete description of each defence option is given along with their abilities. It also offers hotkeys for quick detonation

In addition, upon placement, you are shown the range of the defence, which helps a lot with the gameplay

An upgrade bar on the bottom allows you to extend the range of the defenders or increase the damage they cause

These are the famous spikes which were very popular in my interviews. However, they are really cheap and mean that they make the game too easy. It is better to have a high cost to deter use for emergencies only.



There is information on the next wave of attackers to help the user in planning their defence

Kingdom Rush TD has a very good storyline to follow. It explains the entire story behind why towers are being defended and tend to be popular when the story is very interesting.

You are also often offered the chance to sell your tower in order to gain money and then buy a better one in its place.

Enemies also have information on them which is shown to the user to help them with their defence.

Researching on Wikipedia

[https://en.wikipedia.org/wiki/Tower_defense, Accessed 12/09/2017]

“The basic gameplay elements of tower defence are:

- *territories or possessions (or collectively the "base") that must be defended by the player*
- *the base must survive waves of multiple incoming "enemy" attacks*
- *placement of "Tower" elements, or obstructions along the path of attacking enemies”*

“Some features of modern tower defence:

- *Player placed obstructions that can damage or kill enemy attackers before destroying the base*
- *Ability to repair obstructions*
- *Ability to upgrade obstructions*
- *Some sort of currency with which to purchase upgrades and repairs (this can be time, in game currency or experience points, such as being earned by the defeat of an attacking unit*
- *Enemies capable of traversing multiple paths*
- *Each wave usually has a set number and types of enemies”*

According to this Wikipedia article, Tower Defence games don’t tend to be real time but rather be turn based, whereby you have a build, defend and repair stage. This suggests that tower defence games tend to have a set structure that they follow which should also be implemented as part of my game

Problem Specification

The following features will be important in order to create a good tower defence game:

- *The game should be endless until failure*. The reason for this is that during my related games research, I found that games with levels weren't as fun as games without as you would have just begun engaging fully with the game when all of a sudden, the level would be over and you had to start again. By having an endless game, you have the ability to add to your initial setup and progressively improve your defence as the game gets harder with tougher enemies and of a greater quantity.
- *It is necessary for the game simulation to be slowed down or speeded up at times.* This will avoid the user getting bored as they will be able to speed up the game when they feel as though they should be able to win. In addition, it allows careful placement of weapons by slowing down the game and adding weapons as soon as they are needed (or to add emergency weapons). The speed dial was a common feature on most tower defence games.
- *There needs to be a significant increase in difficulty as you progress through the game.* This can come through tougher enemies or more enemies attacking in one wave. My interviews and research showed me that tower defence games fail to entertain when you can power through missions with the same strategy throughout. There needs to be enough discouragement of buying excessive weaponry and a decent increase in difficulty throughout the game for the game to be considered good.
- *Building times need to be implemented.* This is a common feature in tower defence games (although not all games use it) and should help to aid with the balance of the game, avoiding the game being too easy to complete. It was also requested by my interviewees and so overall should be very helpful to add into the game.
- *Emergency Powerups need to be included.* This is to help users during the game in case of the defence not being sufficient, but should have a high enough cost to deter the user from using it.
- *Health bars, information on weapons, ranges of weapons, upgrades of weapons and selling weapons* are all common features that a tower defence game would be incomplete without. This helps make the game a strategy game and aids the user in making their decisions throughout the game. It is therefore essential to have these.
- *There will need to be an in-game currency system and scoring system.* This is used to buy weapons to create a tougher defence and to give the user an incentive to score more points.
- *There needs to be a save file showing all the high scores on the user's PC.* This adds a competitive edge to the game as users will compete to get a greater score than each other. This is a feature not all tower defence games have even though it would prove popular with all users.

- *There needs to be a button to start the attack so that users can prepare beforehand.* This is essential as it is a strategy game and the user must be prepared before the attack starts. Therefore, by having a button to say they are ready for the attack, they are able to create a defence and start playing when they are ready.
- *Menus and clear instructions need to be present.* This enables the user to understand how they need to play the game and where they can find the right information. These menus may need to lead to information on high scores, enemies, weapons etc. as well as any other help the user may require (and of course a button to begin playing a new game). This is essential in all games otherwise the user may not know what to do.
- *Information about the next wave of attackers.* This is available in all good tower defence games and is an important aspect of tower defence. This is because tower defence in general is a strategy related game, and the best way to get the user to use various strategies is to tell them what is coming up next. They can therefore adjust and improve their defence accordingly and be prepared for when they come.

Optional additions include:

- *Storyline* – These help add to the game but are not essential as not everyone reads the storyline. In this game, only an intro should be required as the game is endless.
- *Good clean design with decent graphics* – This was requested in the interviews but was not ranked as highly needed as the other essential features.
- *A variety of maps.* This makes the game more interesting but is not essential as many good games can be created with just a single map. This is something that was asked for in the interviews, but should only be added after the rest of the game has been implemented fully.
- *Hotkeys.* Hotkeys aren't used a lot but are a feature I thought should be absolutely essential in tower defence games. For those who play a lot of tower defence, they may seek to learn a quicker way of deploying weapons, and hotkeys allows them to do that. This will help improve my game but again is not an essential feature.

My stakeholders reading and agreeing to the problem specification outlined above:

Signing off the Proposed Solution Specification:

I, the named stakeholder of this tower defence game, agree and accept the features stated in the proposed solution specification above.

Miron Abhayasinghe _____

Michael Kuc _____

Praveen Murugathas _____

Vinayak Shastri _____

Oliver Wales _____

Nathan Wang _____

Updates to the original specification

My stakeholders had now agreed to the proposed problem specification, and so I was at a stage to continue with my project. However, I first decided to review the problem specification one last time and I came across a few parts within it which I did not think should have been part of the final specification. This meant that I had to edit my problem specification once more and create a new and improved specification.

There was nothing more that needed to be added but a few features had to be removed. These were as follows:

- Telling which enemies are to come in the next wave is a bit pointless for the game. This is because the game enemies are going to be randomly generated at run time and there is no guarantee of what will come next. This randomness is the unique feature of my game and so by telling the user what is to come next, I will be inadvertently removing my main feature. In addition, when the enemies do come, users should be easily able to see what enemy is coming next without the need for some text.
- The game is endless until failure, so building times are useless. This is because after each wave, the user gets the chance to add to their defence and cannot add anything else during the runtime of the game. If the user was allowed to add weapons during runtime, it would have been useful to have building times to avoid misuse of the feature. Therefore, this feature should be removed to avoid contradicting other more essential features.
- Difficulty will instead be increased by increasing speed as this also helps keep the game fun as users don't need to wait longer periods of time as the game progresses.

Problem Specification 2nd Draft

The following features will be important in order to create a good tower defence game:

- *The game should be endless until failure*. The reason for this is that during my related games research, I found that games with levels weren't as fun as games without as you would have just begun engaging fully with the game when all of a sudden, the level would be over and you had to start again. By having an endless game, you have the ability to add to your initial setup and progressively improve your defence as the game gets harder with tougher enemies and of a greater quantity.
- *It is necessary for the game simulation to be slowed down or speeded up at times.* This will avoid the user getting bored as they will be able to speed up the game when they feel as though they should be able to win. In addition, it allows careful placement of weapons by slowing down the game and adding weapons as soon as they are needed (or to add emergency weapons). The speed dial was a common feature on most tower defence games.
- *There needs to be a significant increase in difficulty as you progress through the game.* This can come through tougher enemies or more enemies attacking in one wave. My interviews and research showed me that tower defence games fail to entertain when you can power through missions with the same strategy throughout. There needs to be enough discouragement of buying excessive weaponry and a decent increase in difficulty throughout the game for the game to be considered good. In addition, the enemies will exponentially speed up as the game progresses.
- *Emergency Powerups need to be included.* The power up should only be allowed to be used once and will kill all enemies on screen when used.
- *Health bars, information on weapons, ranges of weapons, upgrades of weapons and selling weapons* are all common features that a tower defence game would be incomplete without. This helps make the game a strategy game and aids the user in making their decisions throughout the game. It is therefore essential to have these.
- *There will need to be an in-game currency system and scoring system.* This is used to buy weapons to create a tougher defence and to give the user an incentive to score more points.
- *There needs to be a high scores file on the user's PC.* This adds a competitive edge to the game as users will compete to get a greater score than each other. This is a feature not all tower defence games have even though it would prove popular with all users.
- *There needs to be a button to start the attack so that users can prepare beforehand.* This is essential as it is a strategy game and the user must be prepared before the attack starts. Therefore, by having a button to say they are ready for the attack, they are able to create a defence and start playing when they are ready.

- *Menus and clear instructions need to be present.* This enables the user to understand how they need to play the game and where they can find the right information. These menus may need to lead to information on high scores, enemies, weapons etc. as well as any other help the user may require (and of course a button to begin playing a new game). This is essential in all games otherwise the user may not know what to do.

Optional additions include:

- *Storyline* – These help add to the game but are not essential as not everyone reads the storyline. In this game, only an intro should be required as the game is endless.
- *Good clean design with decent graphics* – This was requested in the interviews but was not ranked as highly needed as the other essential features.
- *A variety of maps.* This makes the game more interesting but is not essential as many good games can be created with just a single map. This is something that was asked for in the interviews, but should only be added after the rest of the game has been implemented fully.
- *Hotkeys.* Hotkeys aren't used a lot but are a feature I thought should be absolutely essential in tower defence games. For those who play a lot of tower defence, they may seek to learn a quicker way of deploying weapons, and hotkeys allows them to do that. This will help improve my game but again is not an essential feature.

Obviously, some features of the game have therefore had to be removed, and these are as follows:

- Nathan mentioned how silly telling the user the enemies that will be coming in the next wave is. Therefore, there is no longer a bar along the bottom of the game screen which shows which enemies will be coming up in the next wave, but instead the user will be able to see the enemies using their own eyes
- Building times are an unessential feature. Seeing as the game is endless until failure and you are not allowed to add weapons during the game simulation, building times are irrelevant. Therefore, there will be no building times in the final game.
- Difficulty will now be increased through slowly increasing enemy speed in addition to the previous method. This is because otherwise, a user may be able to find a weapon setup which kills all enemies every single time making the game not as fun to play.

Stakeholder response:

5 thoughts on "Updates To The Problem Specification"

Nathan
NOVEMBER 21, 2017 AT 3:10 PM ~ EDIT

I am forced (by my conscience) to agree to these changes.

★ Liked by 1 person

Reply

Michael Kuc
NOVEMBER 21, 2017 AT 3:13 PM ~ EDIT

I agree with many of these requirements, especially:
The game should be endless.
The game should time control – be careful about this however, as this could add a lot of complexity to the solution.
There should be a high-scores file.
There should be waves of enemies.

However, prioritise these features – not all are as important as the other requirements.

★ Like

Reply

Praveen Murugathas
NOVEMBER 21, 2017 AT 6:05 PM ~ EDIT

I agree!

★ Like

Reply

Miron
NOVEMBER 22, 2017 AT 6:03 PM ~ EDIT

I agree on the updates to the problem specification

★ Like

Reply

Vinayak Shastri
DECEMBER 15, 2017 AT 6:53 AM ~ EDIT

good job

★ Like

Reply

Limitations of the game

The proposed solution is not perfect, and there are some parts of the solution which means that a game suitable for the entire market is not being created.

1. Each tower defence game player has different specifications which they want, which is impossible to have in the same game. For example, while one user prefers levels in a game, anther prefers the game to be an endless type of game. This means that not all of my users will be 100% satisfied with the game as it doesn't completely suit their needs. However, as it is only possible to have one kind of game – levelled or endless – this is a limitation of my solution.
2. The proposed solution is a very big task to handle in a short amount of time. Though most of my key features should be covered in the time given, it will be very difficult to add other important features which quite often influence a user's decision to play a game in the first place. Graphics are key, but they take a lot of time to develop, and so this may mean that the game is not as popular as other games. However, I may be able to overcome this problem by making it a "simplistic" game which has "ok" graphics which works well with the casual user. This does depend on how much time I have left at the end of the implementation.
3. Tower defence games struggle the most with balance. The games tend to have a clear strategy to win the game and have the ability to power through all the levels without a problem. This could be a problem I have in my game as well, or I could even end up making the game too difficult to complete.

Software and Hardware Configurations:

The game should be able to run on most modern computers and laptops, and will be designed for any Operating System after and including Windows 7. There should be at least 1GB of RAM to run the operations of the game, and have around 200MB worth of free disk space. There are no other major requirements, but the game will not necessarily be limited to these specifications.

In terms of software, the computer will need to be able to run C++ and the SFML libraries which should come included with the software and with the operating system (given that it is a Windows Operating System).

Success Criteria for my game:

1) The game needs to be entertaining. Users should be able to play for the times they have stated in their interviews (an average of half an hour) and not get bored in that duration. If they feel as though the game is being repetitive and is not allowing them to engage too well, it is not a good tower defence game. Therefore, this is an important factor in determining whether or not the game is successful. I should expect users to play for about 15-30 minutes based on the research I have carried out.
2) The game should run fairly smoothly. It isn't too uncommon for games to lag a little during run time, but normally these aren't too visible. If in the duration of a gameplay there is significant lag, it might be deemed a failure. For the programming of the game to be considered successful, the game should run smoothly which will reflect efficiency in code, and so this is a good factor.
3) Users should feel that their issues in terms of the problems usually faced in tower defence problems have been addressed. The biggest problem needs to be that users should not be able to power through all the levels and equally not be facing excessive difficulty. They will need to be quizzed on the "balance" of this tower defence game. The balance of the tower defence game cropped up in my interview session time and time again and so will be an important factor in determining the success of my game.

Appendix A – Interview Session 1

Summary of the interviews carried out in the first session

Interview 1 Questions – Michael Kuc - 15/09/2017 - 14:20 PM

How much time do you roughly aim to spend on a mission/game?
30 minutes on tower defence, 15 minutes to an hour
Should the storyline be fully integrated into every part of my game?
I think storylines are less important than the game mechanics. However, once the mechanics are completed, if there is any time remaining, it would be good to have a storyline.
Should a mission last longer than a minute, or should there be speed options to help with this.
Intentional limits on upgradability. Tower defence game levels should start off difficult but become easier towards the end.
In terms of graphics, how important would you rate them to be in order to help make the game more addictive?
Not as important as the game mechanics and gameplay itself
What kind of features are missing from modern day tower defence?
Be able to upgrade indefinitely. You should have building times for towers to force you to think about the future.
Should the power up be one time use only or multiple uses?
It would be better to have tower types than to have power ups. One use.
Should the user be allowed to modify their defence during the simulation?
You should be allowed to add stuff while the game is running. This makes the user more involved as instead of just looking at the screen realising defeat, they can react to it as well.

Interview 1 Questions – Miron Abhayasinghe – 18/09/2017 – 13:40 PM

How much time do you roughly aim to spend on a mission/game?
30 Minutes
Should the storyline be fully integrated into every part of my game?
Storyline is not relevant. I just press the skip button anyway.
Should a mission last longer than a minute, or should there be speed options to help with this.
The game can become repetitive very quick so having speed options would allow the game to not become boring as quickly
In terms of graphics, how important would you rate them to be in order to help make the game more addictive?
Graphics are really important for this kind of game
What kind of features are missing from modern day tower defence?
Levels with different objectives rather than going ahead and kill them e.g. “Kill ten of these things with a rocket launcher”
Should the power up be one time use only or multiple uses?
You could have one powerup in the entire mission to avoid the game becoming too easy and overpowered. Maybe have reset times for the powerup.
Should the user be allowed to modify their defence during the simulation?
You should not be allowed as it makes the game too easy and avoids the strategy side of the game.

Interview 1 Questions – Nathan Wang – 15/09/2017 – 14:00 PM

How much time do you roughly aim to spend on a mission/game?
15 to 30 minutes
Should the storyline be fully integrated into every part of my game?
They are not really important unless they are deeply integrated. Normally not everyone cares about the storyline, they just play it for the fun of it.
Should a mission last longer than a minute, or should there be speed options to help with this.
They can become repetitive over time and you end up just sitting there so speed options would be useful in this sense
In terms of graphics, how important would you rate them to be in order to help make the game more addictive?
The game mechanics are more important than the graphics (and associated things) because if it looks nice but isn't a good game, then what is the point?
What kind of features are missing from modern day tower defence?
Something to get the user involved during a round, such as a spike in Bloons tower defence. You should be able to aim with a mouse.
Should the power up be one time use only or multiple uses?
One time use. There should be a balance as there can be a ton of power ups but that could make it unfair, and some have hardly any (if any at all). The power ups shouldn't break or ruin the game.
Should the user be allowed to modify their defence during the simulation?
Yes but with some sort of delay in building times, emergency things should be allowed but the user should be discouraged from using these.

Interview 1 Questions – Oliver Wales – 15/09/2017 – 08:10 AM

How much time do you roughly aim to spend on a mission/game?
Any game – 1 hour tower defence game – 1 hour as you can't save your progress
Should the storyline be fully integrated into every part of my game?
Don't really care about story lines as long as the game is good and balanced. It should be a challenging game but not impossible – keeps the game interesting.
Should a mission last longer than a minute, or should there be speed options to help with this.
It can take ages at the later levels and it can become a big mess as everyone begins to die. Later levels have thousands of units which also takes ages for the level to finish so speed options are something that is key to the game. Without them, the game would be way too long and boring.
In terms of graphics, how important would you rate them to be in order to help make the game more addictive?
The strategy is more important – there should be multiple strategies that you can play your own way
What kind of features are missing from modern day tower defence?
Currently the enemies all follow one route – have more than one route. The attackers would go through the safest place.
Should the power up be one time use only or multiple uses?
Power ups are good and should be limited in use. One game I played offered a nuke to kill all enemies if it gets too difficult but because it has a reset time, it doesn't ruin the balance of the game.
Should the user be allowed to modify their defence during the simulation?
Not turrets but emergency defences such as spikes. These should also be really expensive to put people off from buying them unless it's absolutely urgent. This makes the game more interesting.

Interview 1 Questions – Praveen Murugathas – 14/09/2017 – 13:30 PM

How much time do you roughly aim to spend on a mission/game?
30 Minutes
Should the storyline be fully integrated into every part of my game?
Quite a lot of detail should be put in. The storyline is the most interesting part of the game and without the storyline, it wouldn't be as good.
Should a mission last longer than a minute, or should there be speed options to help with this.
Near the end, when you cannot complete the mission and begin to lose hope, it can become dull. You should implement changing speed to make it go quicker and not make it silly and boring closer to the end of the game. Always have an end to the game, especially with the storyline.
In terms of graphics, how important would you rate them to be in order to help make the game more addictive?
Not really, a simple User Interface is okay.
What kind of features are missing from modern day tower defence?
Make it use 3D Graphics and make it non-classical
Should the power up be one time use only or multiple uses?
One time use to make the game interesting – it really adds to the experience
Should the user be allowed to modify their defence during the simulation?
Playing during the game would ruin the game.

Interview 1 Questions – Vinayak Shastri – 18/09/2017 – 13:30 PM

How much time do you roughly aim to spend on a mission/game?
Usually about an hour but I do two hours a day of gaming
Should the storyline be fully integrated into every part of my game?
I really enjoy storylines as long as we are allowed to deviate from it
Should a mission last longer than a minute, or should there be speed options to help with this.
Tower defence as a genre is not that fun so it gets boring after a while. The mission time should be reduced as a result.
In terms of graphics, how important would you rate them to be in order to help make the game more addictive?
The Graphics should match with the gameplay and action going on within the game itself. I.e. Simplistic motion should be match with simplistic graphics.
What kind of features are missing from modern day tower defence?
I cannot think of any features that would help add to the genre of tower defence
Should the power up be one time use only or multiple uses?
The power ups should be one time use only to void making the game unbalanced
Should the user be allowed to modify their defence during the simulation?
When the gameplay commences, there should be no further input to make it a more tactical game

Tower Defence Game Design
Ghusharib Chohan 605

Decomposition of the problem:
Making a game is a very complex procedure that requires pages and pages of code to be made effective. Fortunately, as games are very structured and linear, it is fairly easy to make as you can code parts of the game at the time in a modular approach. The modules can be broken down as follows:

The game module – This module is more of a main module where all the code is being run. As the game runs sequentially, I will require the different modules of code to run one after the other in a structured order. I will also be required to find out when the user wants to end the game and as such close the game in a safe manner upon request. All of this should be contained in one module.

Simulation Module – One of the modules needs to be dedicated to simulating the running of the animations of the game. This needs to be done in a simulation module as each animation needs to run one after the other. The module will need to get the correct object and change its photo to the next image at the correct time to produce an animation.

Updating the objects – Another Module is required to update the values of objects running in the game. For example, when an enemy is attacked, their health needs to go down and be registered as being hit. This will therefore be updated by the module so that the object's health can be decreased accordingly for every enemy that is hit.

Updating the currency – When the user earns in game money, this needs to be registered and outputted as a display to the user so that they know how much money they have available to spend. Equally, the module of code should be able to decrease the amount of money available when it has been spent. Should they not have enough money for a transaction, an error message will need to be shown.

Updating the score – Every time an enemy is killed, a running total of the score will need to be updated dependent on the type of enemy killed. This will also then have to update the user interface to show the current score and make the user aware.

Save high score module – The game is endless until failure, so a save file is not necessary. However, it would be very useful to have a file with all the high scores contained in it for the user's reference. This will be required to be updated, added to and outputted on the user screen upon request and therefore requires a module of its own.

Menu module – When the user clicks on various buttons on the menu, it should lead them to the correct part of the menu that they are looking for e.g. High scores, Info, Help etc. These will then need to be further developed into smaller sub modules so that they do the job they are supposed to do.

General Update for the user interface – A module of code is required to update the general details of the user interface. This includes highlighting the next wave of attackers and all the necessary details for the user.

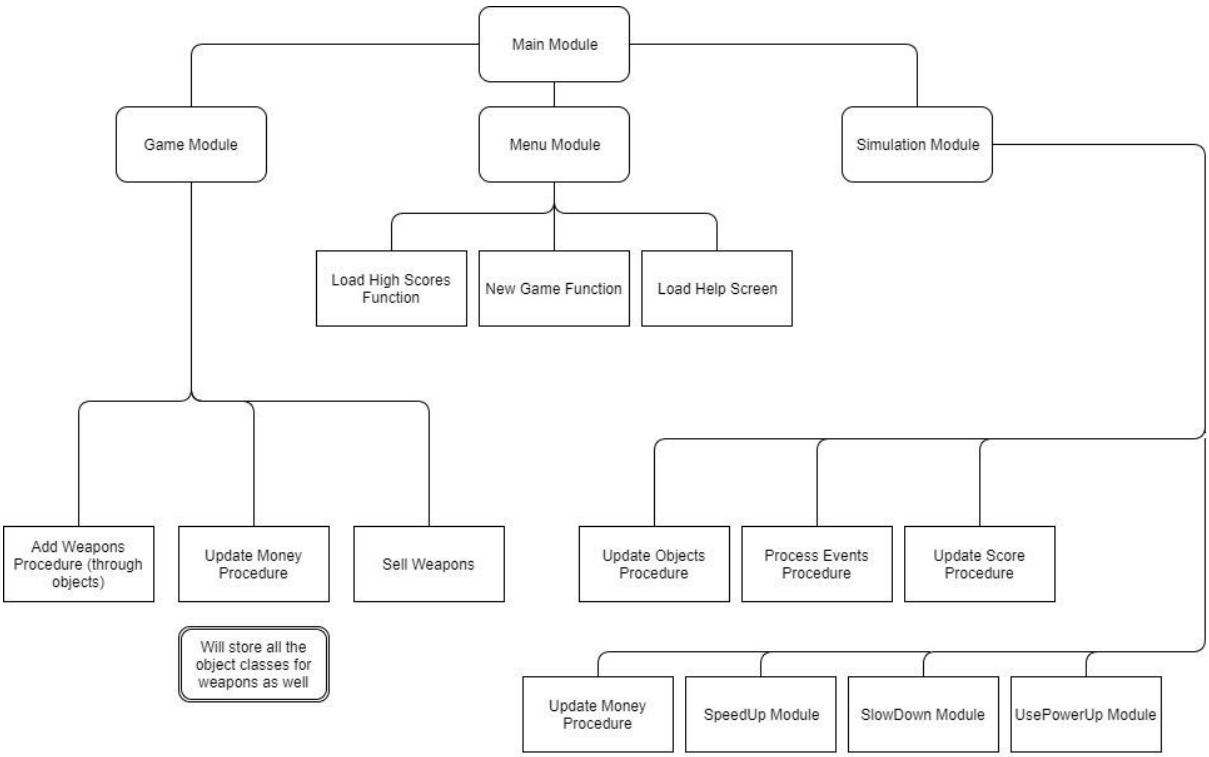
Speed module – A module of code should be able to increase the simulation speed and so will be closely linked to the simulation module.

Select and drag module – The user needs to be able to select the weapon they wish to purchase and then drag it to the right position on the grid that they wish to place it. This requires a module of its own to update the object's values.

Add/Delete new object – Every time a weapon is purchased, a new object will need to be created using the correct class which requires its own module. Equally, should the user sell their weapon, this module needs to remove the object.

Diagram to show the modules of code

The relationships between these modules and what they will lead on to do are best shown in a diagram. This is the decomposition/stepwise refinement of the problem and how it will essentially follow through modules of code. By programming the most basic parts of the refined problems shown below, my programming task will be made far, far easier.



How these modules fit together to form a complete solution

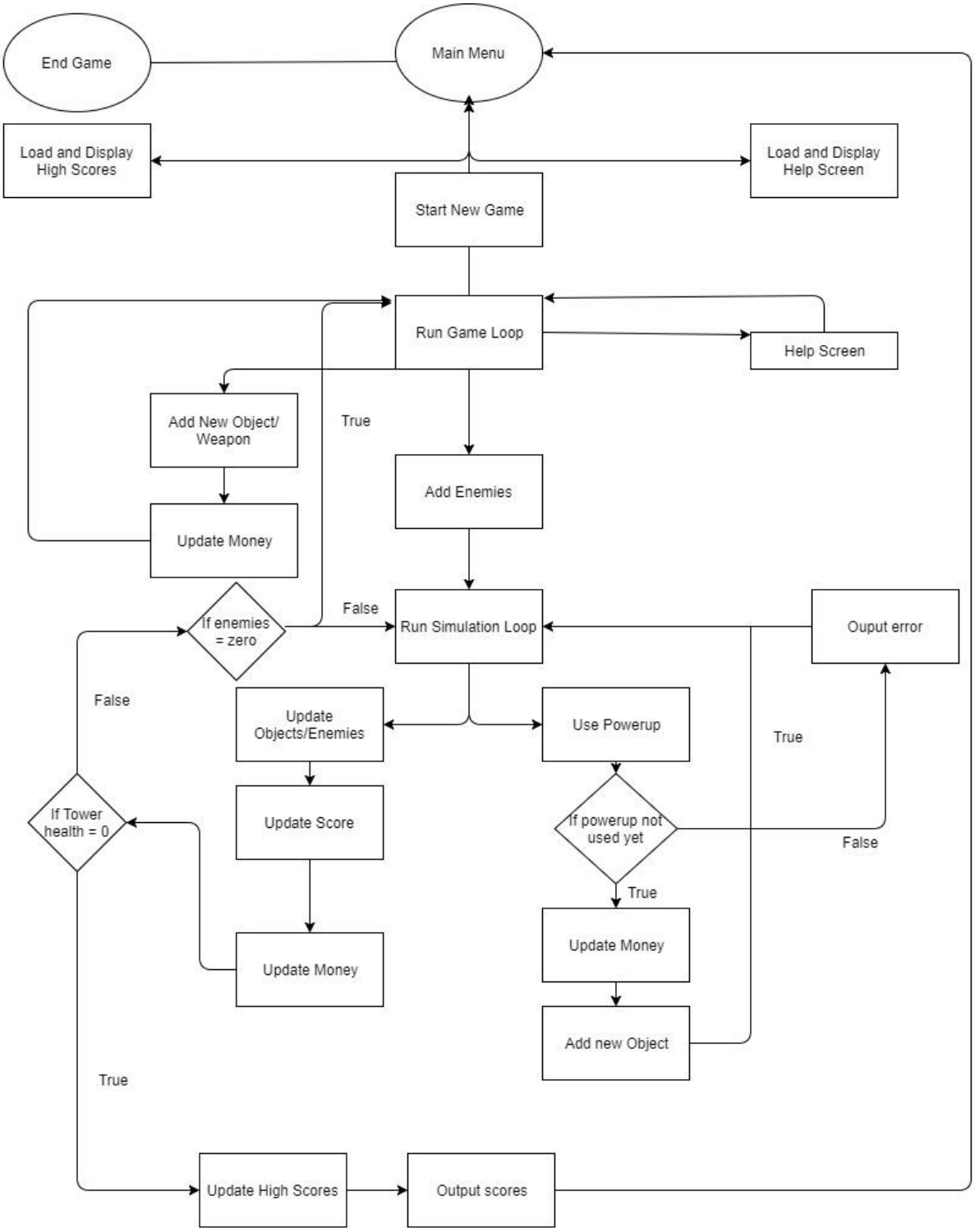
The modules of code shown in the page before all combine to form a solution to the problem – together they form a fully functioning game. For example, in the Pre-Game Section, the Modules for Buying Weapons, Updating Money and Selling Weapons all allows for the user to build a defence in order to protect themselves from the attack.

The Simulation Module is subdivided into smaller modules. A module to update objects ensures that all objects on the screen are moved accordingly e.g. Enemies move along the path and weapons follow enemies. The Update Score module ensures score and money is updated as the game is played and the Speed Up And Speed Down Modules allow for the game to be sped up or slowed down accordingly. All of these modules work perfectly well on their own and so are self-contained as is the requirement for all modules of code. However, on top of that, when these modules are combined, they form the entire basis for the simulation.

Similarly, the Load High Scores, New Game and Load Help modules all form part of a Menu to allow Menu Like Functionality. These three bigger categories (which I have divided into sub-categories) are the basic needs of every game – A Menu to start the game and get help from, a pre-game loop to allow the user to build a defence and a simulation module to run the actual game. As a result, these three categories combine together to form a full and complete solution that is much more easily programmable.

The process of how the game will run through the form of a flowchart:

To best way to understand the algorithms, classes and objects (as well as any other data structures) that I may need to implement, is to first understand how my game should flow and the steps my game will go through. This should help me formulate a plan of attack in terms of how the modules of code I came up with previously can be created. The flowchart follows a similar structure to the module relationship diagram shown on the previous page:



This structure is a good structure for a game:

When the user starts, they will need a Main Menu to be present so that they can navigate their way through the game and access other features if they need to (rather than diving straight into the game). This main menu needs to be kept simple so that users don't look at it and get confused by it. Therefore, three options from the main menu should be fine.

The High Scores and Help Screen options can be accessed from the main menu and are standalone processes (as in they do not lead on to another process). This enables the user to simply load the information they want and once they are done, they are able to return to the main menu as when they click on these options, they only expect to see the information they want and return to where they started. The third option is to start a new game which will put them into the game loop. This game loop is necessary to keep the game running in time and on track.

The game loop has two potential loops. One is the help screen again. This is because the user may not have necessarily looked at the help screen when given the option to on the main menu and may still need help. To avoid them having to unnecessarily end the game and return to the main menu just for help, a help screen in the game loop will help them get help even during gameplay. The second potential loop is when they add a new weapon. This is because the entire point of the game is to place weapons to help your defence. This loop will add the new object in the position that the user drags the object to and will then update (deduct) money from their funds. It is important to deduct money to ensure that the user is not able to spend more money than they actually have.

Once the "play" button has been pressed, the Game Loop is broken out of, and then enemies have to be added. This is to ensure that the simulation can run and that there is something attacking the user (otherwise the game is pointless). The game will then enter the second loop – the simulation loop. Once again, this has two potential loops. One loop comes from if they decide to use a power up. If this happens, a logical decision will need to be made. If a power up has not yet been used, they will be allowed to buy one and so money will be deducted and the power up object will be added. However, if they have used a power up already, they will be given an error message in words to that effect. This will then loop back into the simulation loop.

In all other events, all the objects on the map will need to move certain places, the weapons will attack the enemies if they are in a certain range and as such the enemy health (or the tower health) will decrease as a result. This will all be handled by the Update Object/Enemies module and is an important feature otherwise the game will not function properly. This will then need to be followed by an update in the score and money as for all damage to the enemy, the user will gain some score and

earn money from it. From these updates, a logical decision then needs to be made again.

If the health of the tower is 0, then the user has lost the game. Therefore, if they made a new high score, the high scores will need to be updated and in all cases, the score they achieved will need to be output on the screen. The user should then be taken back to the main menu to decide what they want to do from there.

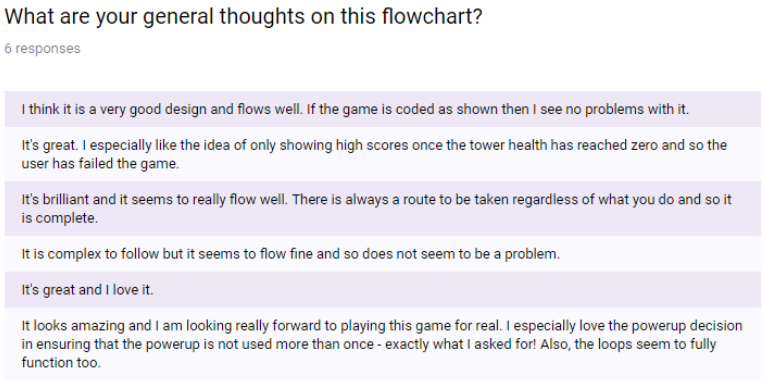
However, if the health of the tower is not 0, then the user has not yet lost. Therefore, another logical decision needs to be made in terms of whether or not all the enemies have died. If they haven't, the simulation loop will just loop again, but if they have, then the user has passed this mission. The game needs to then therefore go back to the game loop to allow the user to add more weapons using the money they earned and hence improve their defence (and so making it a strategy game). Without these checks, the game may never end.

From the main menu, there is always an option available to end the game so that the user can quit and do something other than play the game.

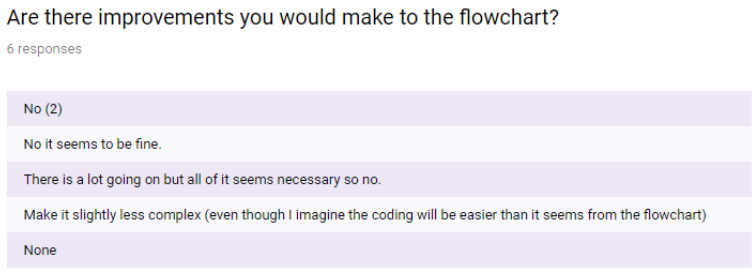
User feedback for flowchart

For the flow-chart, I decided to get feedback from my users as this was essentially what the code would look like and how the game would flow. The questions I asked were as follows:

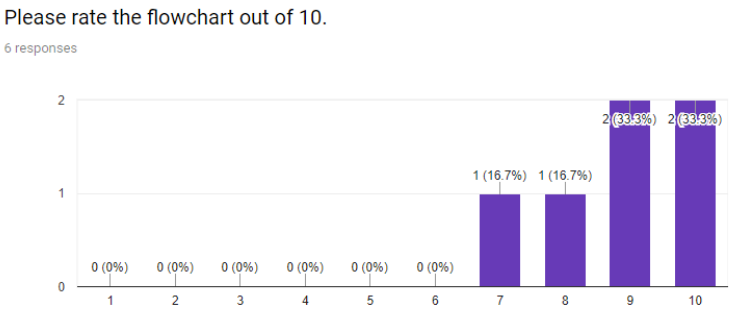
1. What are your general thoughts on this flowchart? This is open enough to allow the users to give sufficient feedback on what they genuinely think about the flow
2. Are there improvements you would make to the flowchart? This allows the user to have some input on the flow of the game.
3. Rate the flowchart out of 10 – this gives a quick summary of the general consensus of my stakeholders.



My stakeholders were generally happy with the flowchart, although one mentioned it seemed to look really complex!



Only one user thought it could be improved and that was to make it less complex. However, as pointed out, this should be fixed when it comes to coding.



The average rating for the flowchart was a healthy 8.8 showing the users were generally very happy with the flow of the game.

As I have the green light from my stakeholders implying that the game design seems to flow well, I can concentrate on some further adjustments before concentrating on the development of the game.

Pseudocode

Enemy type algorithm - One of the new additions I am making to my tower defence game is that every time someone plays the game, a random game is created. This is done by creating a random enemy line up every time the game is played, and so some pseudocode needed to be made before I could implement it (and hence it is part of the design stage)

typeOfEnemiesToAdd()

```
1  // Score would start above 0, Assume an initial score of 50
2  // Enemy 1 has score 10, 2 has score 50, 3 has score 100
3  PROCEDURE typeOfEnemyToAdd(int Score)
4      IF (Score > 50)
5          WHILE (Score != 0)   // Once Score is 0, algorithm cannot add more enemies
6          int n = rand(1,3)  // Enemy addition needs to be random
7          SWITCH (n):
8              CASE (1):
9                  addEnemy(1)
10                 Score = Score - 10
11                 // The first kind of enemy can always be added given score > 0
12                 // In which case the enemy is added and the score is reduced
13                 BREAK
14             CASE (2):
15                 IF (Score - 50 >= 0) THEN
16                     // If the enemy can be added without the score going below 0
17                     addEnemy(2)
18                     Score = Score - 50
19                     BREAK
20                     // In which case the enemy is added and the score is reduced
21                 END IF
22             CASE (3):
23                 IF (Score - 100 >= 0) THEN
24                     // only add if it can be added without going below 0
25                     addEnemy(3)
26                     Score = Score - 100
27                     BREAK
28                     // In which case add the enemy and reduce score
29                 END IF
30         END SWITCH
31         WEND
32     ENDIF
33 END PROCEDURE
```

Enemy Type Algorithm – Dry Run

Start with score - 250

Line Number	Score	n	addEnemy()	Score > 0
4	250			
6		3		
25			3	
26	150			
31				TRUE
6		1		
9			1	
10	140			
31				TRUE
6		2		
17			2	
18	90			
31				TRUE
6		1		
9			1	
10	80			
31				TRUE
6		3		
28			Cannot Add	
31				TRUE
6		2		
17			2	
18	30			

31				TRUE
6		2		
17			Cannot Add	
31				TRUE
6		1		
9			1	
10	20			
31				TRUE
6		1		
9			1	
10	10			
31				TRUE
6		1		
9			1	
10	0			
31				FALSE

End of code. It added an Enemy 3, 1, 2, 1, 2, 1, 1 and 1.

Start with score – 3

Line Number	Score	n	addEnemy()	Score > 0
5	3			
6		3		
25			Cannot Add	
31				TRUE
5	3			
6		2		
25			Cannot Add	
31				TRUE
5	3			
6		3		
25			Cannot Add	
31				TRUE
5	3			
6		1		
25			Cannot Add	
31				TRUE
5	3			
6		1		
25			Cannot Add	
31				TRUE

This case would cause an infinite loop. However, this should be fine as the game only deals with scores in multiples of 10 and so is an impossible scenario to occur unless the user uses a cheat engine in an attempt to cheat and edit their score.

Weapon Lock Algorithm – When a weapon fires, it keeps firing on a certain enemy until it ends up outside of its range. This means that there needs to be a function to lock on to a certain enemy until they either die or leave the weapon's radius. Therefore, I created some pseudocode to find the enemy that needs to be locked onto.

LockOnEnemy()

```
1  //Assume Global Vector containing all Enemies objects called Enemies[]
2
3  PROCEDURE LockOnEnemy(int totalEnemies, int ~~totalWeapons~~Number)
4      FOR i IN RANGE (0, totalEnemies -1)
5      // For every enemy in the Vector carry out the following
6          IF isInRange(Enemy.Position(), weapon.Position(), weapon.radius) THEN
7          // If the enemy is in range, lock onto it
8          RETURN i
9          // Returns the enemy number to lock onto
10         END IF
11     NEXT i
12 END PROCEDURE
13
```

This also led to the production of an algorithm to check if a weapon is in range which would be fed into by the weapon lock algorithm. This would use Pythagoras to compare the distance of the enemy from the weapon with the radius of the weapon, and if it was within range, it would only then tell the weapon lock algorithm to lock onto the weapon.

isInRange()

```
1  PROCEDURE isInRange(EnemyPosition, WeaponPosition, Radius)
2      int dx = WeaponPosition.x - EnemyPosition.x
3      int dy = WeaponPosition.y - EnemyPosition.y
4      // dx & dy are the difference in the x and y positions of the enemy and weapon
5      int squared = (dx^2) + (dy^2)
6      int squareroot = squared^0.5
7      // Uses pythagoras to work out distance between enemy and weapon
8      IF (squareroot > Radius) THEN
9          RETURN FALSE
10         // If the distance is greater than the radius, it is out of range
11     ELSE
12         RETURN TRUE
13         // Otherwise it is not
14     END IF
15 END PROCEDURE
```

Weapon Lock Algorithm – Dry Run

Case:

Enemy Number	x	y
0	140	600
1	60	200
2	30	40

Weapon 1 Position – (40, 60)

Weapon 2 Position – (80, 60)

Radius of 100 each

Line No.	i	totalEnemies	weaponNumber	Weapons[weaponNumber].radius > Enemies[i].position?	Return
3		3	1		
4	0				
6				FALSE	
4	1				
6				FALSE	
4	2				
6				TRUE	
8					2

3		3	2		
4	0				
6				FALSE	
4	1				

6				FALSE	
4	2				
6				TRUE	
8					2

isInRange() fed into by LockOnEnemy() algorithm

Line	dx	dy	(dx^2 + dy^2)	Sqrt(dx^2 – dy^2)	Return
2	100				
3		540			
5			301600		
6				549	
9					FALSE
2	20				
3		140			
5			19600		
6				140	
9					FALSE
2	-10				
3		20			
5			500		
6				22	
12					TRUE
2	60				
3		540			
5			295200		
6				543	
9					FALSE
2	-20				
3		140			
5			20000		

6				141	
9					FALSE
2	50				
3		20			
5			2900		
6				53	
12					TRUE

This works as expected. The two weapons both choose to lock onto Enemy Number 3 and no other enemy as this is the only enemy within their range.

Case:

Enemy Number	x	y
0	60	20
1	500	200
2	45	20
3	250	20

Weapon Position – (40, 20)

Line No.	i	totalEnemies	weaponNumber	Weapons[weaponNumber].radius > Enemies[i].position?	Return
3		4	1		
4	0				
6				TRUE	
8					0
4	1				
6				FALSE	
4	2				
6				TRUE	
8					5
4	3				
6				FALSE	

isInRange() fed into by LockOnEnemy() algorithm

Line	dx	dy	(dx^2 + dy^2)	Sqrt(dx^2 – dy^2)	Return
2	20				
3		0			
5			400		
6				20	
12					TRUE
2	460				
3		180			
5			244000		
6				493	
9					FALSE
2	5				
3		0			
5			25		
6				5	
12					TRUE
2	210				
3		0			
5			44100		
6				210	
9					FALSE

This does not work as expected. This is because the algorithm finds two potential Enemies to lock onto, and instead locks onto the enemy that is furthest away. This can be fixed by putting in a break case on line 9 and so the weapon only locks onto the first enemy within its radius.

Update enemy and other values loop – The simulation loop runs throughout the running of the game and simulates/animates the movement of the weapons and enemies. Therefore, pseudocode regarding how the program will use polling to cause movement of the objects was required and so is shown below:

```
PROCEDURE processEvents(int totalWeapons, int totalEnemies)
    FOR i IN RANGE (0, totalWeapons - 1)
    //Carry out the following on every displayed weapon
        IF (weapons[i].weaponReadyToFire) THEN
            weapons.fire()
            enemyLockedOn.reduceHealth()
            //If the weapon is ready to fire, start the firing animation and reduce the health
            //of the enemy that has been locked onto
        END IF
    NEXT i

    FOR i IN RANGE (0, totalEnemies - 1)
        //Carry out the following on every enemy
        Enemies[i].move(speed)
        //Make the enemy animate (and hence move) a certain distance dependent on its speed
    NEXT i
END PROCEDURE
```

This was just a general loop which could not really be tested, and so I decided **not to dry run** this algorithm.

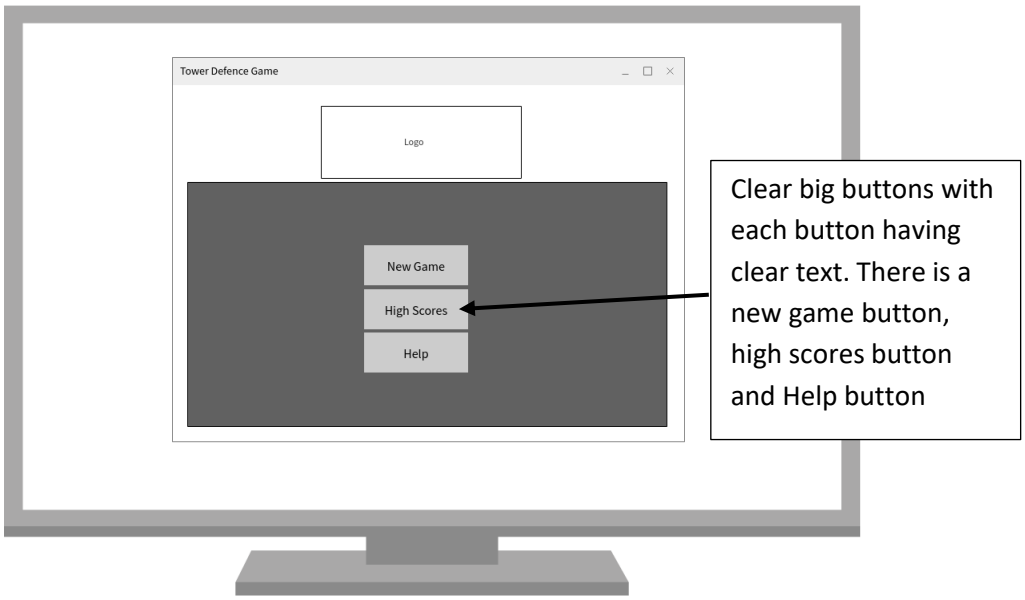
Key Usability Features

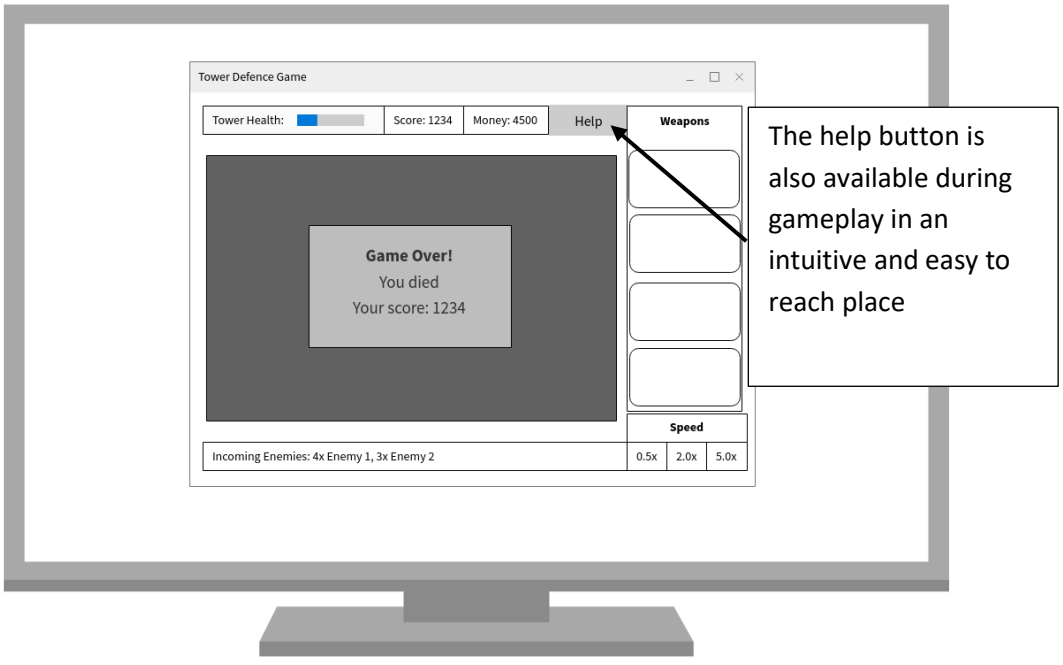
Not all users are experienced as one another – some users require more help than others in terms of how to play games, particularly if this is the first game they have ever played. Therefore, there need to be key usability features involved to ensure that the user knows what they are doing:

- Help screen available at all times – My help button will need to be available at all times, so when the game starts up and when the game is actually running. This help button will load up the help screen so that the user gets the help they require in terms of what the game is about and how to play the game. This will always be available in a visible part of the screen, and so the user will never be left alone without knowing what they should be doing.
- Error textboxes whenever the user makes a mistake – For example, if the user places a weapon on the actual path or outside of the given restrictions, then an error window will open to tell them the error they have made and (if necessary) what they need to do about it (along with a button to close the error screen once again).
- Speed buttons – There should be buttons to increase or decrease the animation speed when the game is running to ensure that the use doesn't have to wait too long when they are happy with their defence (or vice versa). This ensures that the user enjoys the game.
- Clear design with big buttons – this is to ensure that everything that the user needs to play the game is clearly visible to them and they do not struggle to play the game. As a result, the game will not be frustrating for them. This will also be done through clear graphics.

What these usability features should look like:

These are images of a screen mockup whose feedback can be seen later on pages 55-57





Variable, Data Structure and Class Listing

Some variables I know I will need are as follows:

- Enemy Speed – To ensure that the game runs at the same rate on all computers regardless of its specifications (particularly its clock speed), I will require an enemy speed. By keeping it a constant, it ensures that the objects move the same distance per second (regardless if it only travels this rate two seconds later). This is a integer as it stores a number.
- Total Weapons – A variable storing the value of the total number of weapons the user has added needs to be available. This is to ensure that the simulation loop is able to use polling to go through them all and hence simulate the firing of the weapon (if needed). This is an integer as you can only have whole weapons, not halves
- Total Enemies – A variable storing the value of the total number of enemies needs to be available as well so that each enemy can be worked with individually. These two variables are necessary as we cannot know before how many enemies or weapons before the game begins, and therefore a variable can overcome this problem. This is also an integer for similar reasons.
- Tower Health – As the game will only end once the tower's health has reached zero, it is important to have a variable keeping track of the tower's health so that once the game is required to end, the health of the tower can be checked and ended as a result. This is an integer as it is a number that is constantly being added to/subtracted from.
- Enemy Health – Similarly, the entire game runs on the basis that the enemies are attacked by the user's weapons and once their health reaches zero, they are removed and the relevant score and money is added to the user's variables. This is also an integer for the same reasons.
- Weapon Recharge Time – This is required as each weapon is different and so has different features such as recharge time: the time the user must wait before it can be used again. This is an object of the SFML Time class as it is a specific of the SFML Library
- Weapon Damage – Similarly, different weapons will do different damage to enemies, and so a variable storing the damage done by a certain weapon will need to be present. This is an integer.
- Money – A money variable is required to keep track of how much the user has available to spend on weapons and can be updated when weapons are bought or sold as well as when enemies are destroyed. This is an integer.
- Score – This is to keep track of how much score a user has earned so that there is an incentive to do better the next time they play. This is updated every time an enemy is killed. This is an integer.

- WeaponReturnMoney – When a weapon is sold, money will be returned to the user. However, each weapon will return a different amount when it is sold so a variable is required to store this value for the different weapons. This is an integer as well.
- EnemyScore – Each enemy will be scored/ranked according to how tough they are to defeat, and so a variable is required for each enemy (an integer)
- EnemyGainMoney – When an enemy dies, a variable needs to be accessed stating the amount of money the user earns as a result of destroying that particular enemy. (An integer)

Data structures I will need to use

- Vector to store multiple objects – Vectors are essentially dynamic arrays and are very useful for my project. This is because in theory, my game will need to hold an infinite amount of weapon and enemy objects rather than a predetermined number of weapons/enemies. Whenever you create an object, you need to give it a unique name, but vectors overcomes this problem as all you lost the unique name when you store it into a vector and instead you get an indexed value such as vectorName[1] will access the second object in the vector. In addition, vectors are dynamic so I can in theory hold an infinite amount of objects without worrying about having reserved positions in memory.
- Classes – The most important part of my game will be the classes I will use. The classes will define specific features of the game of different types of enemy and weapons, such as the damage they do, the health, reload times and more. Classes are key to this as the user will have multiple numbers of objects relating to their weapons and enemies and the easiest way to give a definition of an entity with multiple variables (without creating individual variables) is by using classes to define these unique variables.

Classes I will need to make

- Enemy (1/2/3) – For each of the enemies, (Enemy1, Enemy2 etc.) there needs to be a different class defining its features. One type of enemy will have a different score to another enemy. These enemy classes, however, will all inherently have the same variables within them:
    - Health variable to store their health so that when it reaches zero, the enemy is dead
    - MoneyGained variable to store the value of the money that will be added to the user's total money once the enemy has been killed
    - Enemy Score variable relating to the score the player will earn when the enemy is killed

  In addition, some methods for the enemy class that may be useful are:
    - updateHealth() to decrease the health when an object is hit by a weapon

- getMoney() to keep the money variable private and have a get function to return the value of money
- getScore() to return the value of the score

There will also have to be some external object destroyer function so that when the health is zero, the object is removed from the screen

- Weapon (1/2/3) – For each of the different kind of weapon (Weapon1, Weapon2 etc.), there also needs to be different classes defining its features. Once again, these classes should have similar variables regardless of which weapon they are:
  - Weapon Reset Time as more damaging weapons should not be able to fire as often as a less damaging weapon and so having a time variable should make this easier to implement
  - Reduce Health By variable to store the health value which when the weapon is fired, it will reduce an enemy's health by. This is as you cannot know which specific weapon will attack which specific enemy before you play.
  - Return Money Variable so that when the weapon is sold, the user can get money back
  - Weapon Location to store the pixel location on the screen as to where the weapon has been placed

  Some methods that may be required within the class are:
  - canFire() function will be required to ensure that the reset time has been achieved before the weapon fires again
  - getPower() to get the value which the enemy's health should be reduced by (accesses the private Reduce Health By variable)

  Like the Enemy class, this will require an external object setter and destroyer so that the display can be updated as required

- Power-up – The powerup is a standalone weapon which requires its own features in terms of variables and methods and therefore it requires its own class as well. The variables for this class will be:
  - isUsed will be a Boolean value ensuring that the powerup is only used once in a mission and not more
  - Reduce Health By variable like in the weapon class to check how much the health should be reduced by for enemies.
  - Location to store the location on the screen where the powerup has been placed

  In addition, some methods will include:
  - canUse() to check if isUsed is true or false and if therefore the powerup can be used
  - getPower() to access the reduce health by variable

Class Diagrams of my classes:

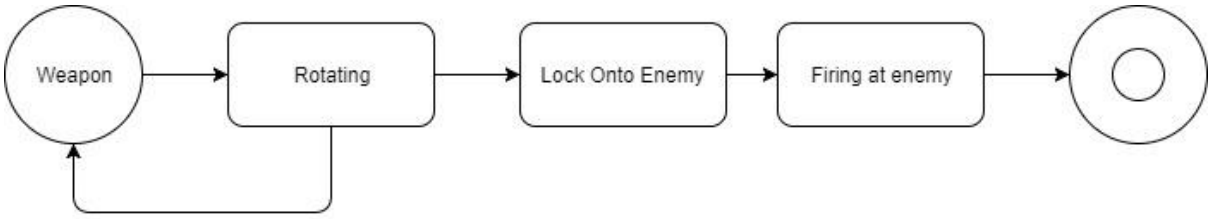
Enemy1
Health Money Gained Enemy Score Enemy Position Enemy Texture
getHealth() getMoneyGained() getScore() getPosition() setTexture() setPosition() updatePosition() updateHealth()

Weapon1
Weapon Reset Time Weapon Damage Return Money Base Position Turret Position Weapon Texture
getDamage() getCurrentTime() getReturnMoney() getBasePosition() getTurretPosition() setBasePosition() setTurretPosition() setBaseTexture() setTurretTexture() rotateTurret() resetTime() updateCurrentTime()

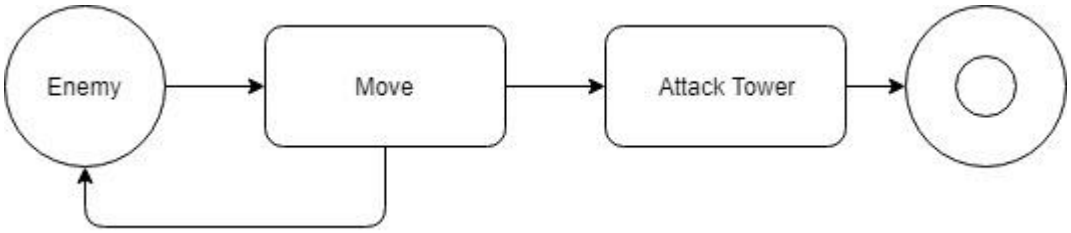
Power-Up
Is Used Weapon Damage Weapon Texture Weapon Position
getDamage() getPosition() setTexture() checkIfUsed() setPosition()

State Diagrams for the classes:

The Weapon State Class

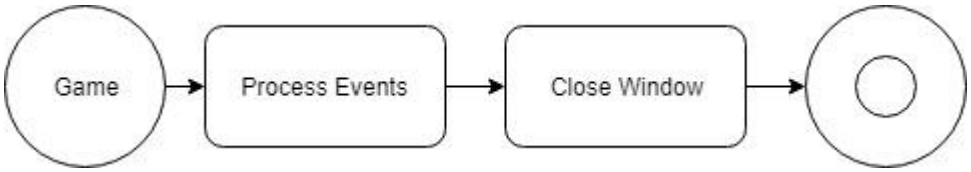


The weapon when in its running state will be rotating, locking onto enemies and firing at enemies.



The enemy on the other hand will only be focussed on moving and attacking the tower throughout its lifetime.

The entire game will also have a state diagram whereby it will be continuously processing events until the window is closed:



Required Validation

As part of the game, there will need to be validation to ensure that the user is carrying out correct procedures. Some of these need to be determined before I begin development:

- Checking if a weapon can be placed where it is being dragged. During the game, the weapon should not be placed too close to another weapon, on-top of another weapon or indeed on-top of the path. If this becomes a possibility, it may ruin the game. Therefore, it is necessary to have the ability to check if the weapon is being placed too close to either of these and if it is, then the correct feedback should be given e.g. an error message saying that this weapon cannot be placed here.
- Mouse and keyboard inputs will require validation in order to check which event to carry out. For example, I will need to be able to differentiate between a mouse click, mouse moved and the various keys on a keyboard. This will help with the save score screen when the user will need to enter their name as well as with the click and drag function for adding weapons to the map.
- Mouse input validation will also be required to check where the mouse is being clicked and if it is being clicked on top of a button (e.g. the pause button), the relevant function should run as a result.
- Checking if an enemy is dead or if the tower's health is below zero. When an enemy's health falls less than or equal to 0, then we will know that the enemy should no longer exist. Therefore, validating the enemy's health value will ensure that the enemy is removed at the right time and the valid score and money is returned to the user as a result. In addition, if the tower's health also needs to be validated, as if its health also falls below or is equal to zero, then we know the game is over and the game over events should start running.
- Comparing high scores. When adding a new high score to the high scores table, I will require validation to compare values already in the high scores database with the new high score value that needs to be put in. If the new score is higher, then its position will be higher (and so the position value be smaller) whereas the opposite is true for a smaller high score. This validation will ensure that the new value added into the high scores database has been put in the correct position.

Specifics of the language and libraries that will be used to program this game

For this game, I have decided that C++ is the most effective programming language to use. This is because C++ is very suitable for windows enabled devices which are the most common operating system on computers across the world. In addition, I know C++ better than other suitable languages such as Python, Java and other high-level languages. Finally, as C++ creates an executable file (in the file format .exe) and can take advantage of static linking (where a library is hardcoded into the executable itself), I will have the benefit of ending with a standalone application that can be run on many computers without the need for additional installations (as most of its requirements will be self-contained or pre-installed on most modern computers).

In terms of library code, I chose the SFML Library (Simple and Fast Multimedia Library) that is made specifically for C++ and gaming. This is because it contains classes and variables which work very efficiently and to a good standard. This means that my game will run very smoothly and quickly. In addition, it is very simple to use compared to other libraries and so it is the clear choice when it comes to game programming.

SFML Specifics which will be beneficial for me:

The sprite class within SFML has many key variables and attributes which will help me with my game development. For example, SpriteName.getPosition() (and the corresponding setPosition() function) is useful as it allows movement of the sprite throughout the game. In addition, it has other features such as setTexture that allow you to use your own images to add to the sprite without much hassle.

User Interface and Feedback

I decided that now I was mostly done with my design, it was necessary to get the next most important part of the design stage – working out what kind of layout was most useful and well placed for my users. I therefore created some mock ups for the user interface screens on mockflow.com and posted these on my blog for my stakeholders to respond to.

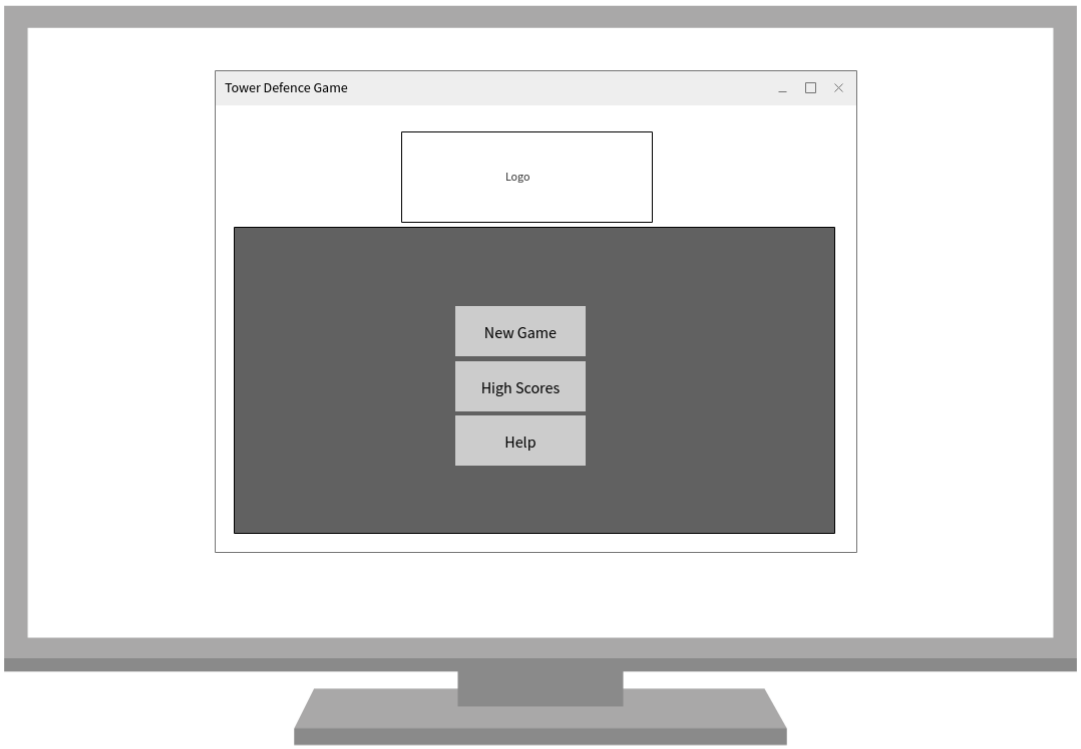
Based on my feedback, I decided on the following:

- The user layout is generally good and doesn't need major improvement
- The actual gameplay area should probably be a bigger feature of the layout and not be controlled to such a small proportion
- On my main menu, I should fill up the empty spaces with some screenshots of the game itself to help with the visual
- A blue colour scheme will probably work best – it is a neutral colour choice and most people's favourite colour

The mock ups and feedback are shown below

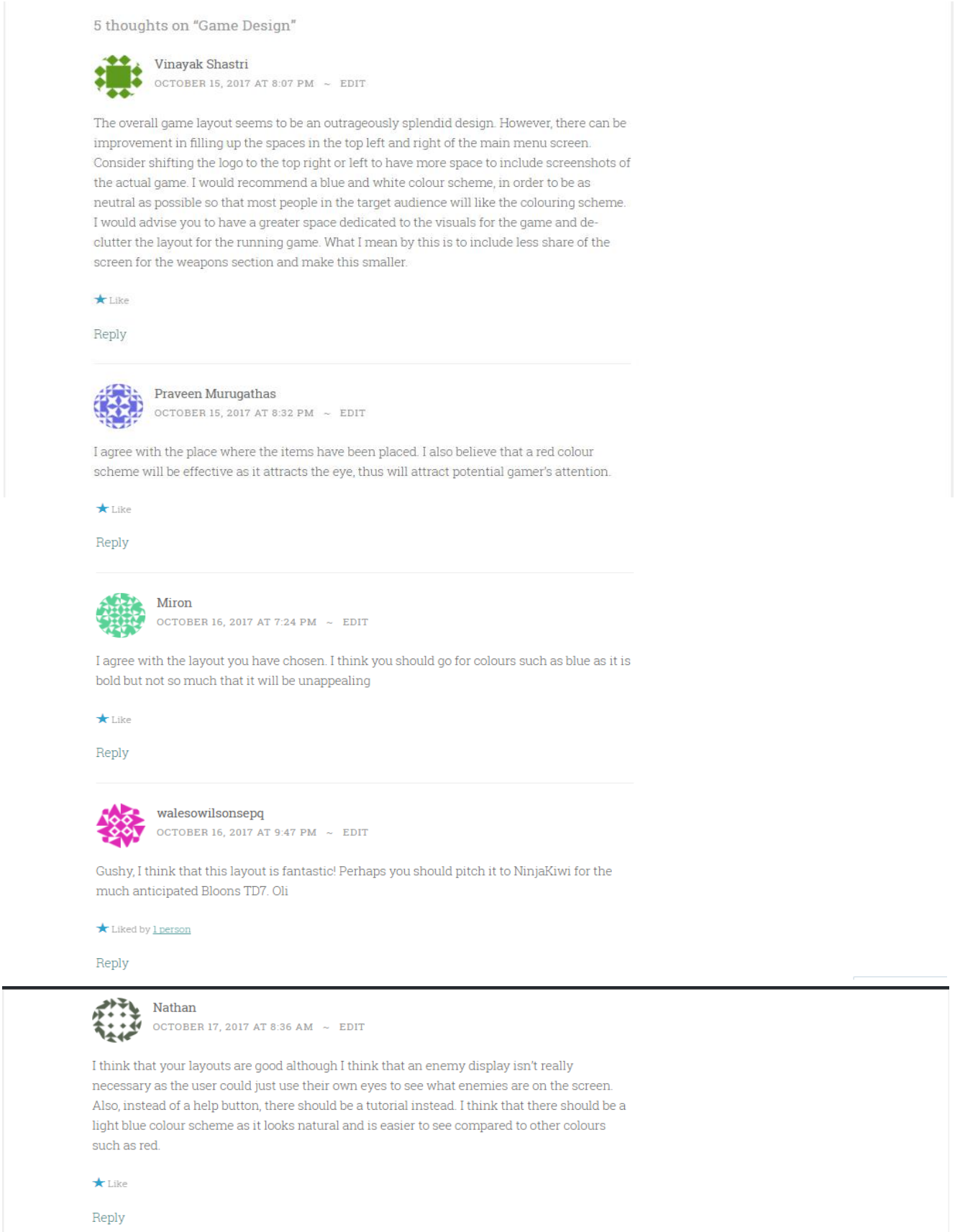
(https://ghusharibcomputinggame.wordpress.com/2017/10/11/game-design/):

Main Menu Screen:



Game Running screens:

I also asked my user's what kind of colour scheme they think would work well with this layout, and this was the response I got:



5 thoughts on "Game Design"

Vinayak Shastri
OCTOBER 15, 2017 AT 8:07 PM ~ EDIT

The overall game layout seems to be an outrageously splendid design. However, there can be improvement in filling up the spaces in the top left and right of the main menu screen. Consider shifting the logo to the top right or left to have more space to include screenshots of the actual game. I would recommend a blue and white colour scheme, in order to be as neutral as possible so that most people in the target audience will like the colouring scheme. I would advise you to have a greater space dedicated to the visuals for the game and de-clutter the layout for the running game. What I mean by this is to include less share of the screen for the weapons section and make this smaller.

★ Like

Reply

Praveen Murugathas
OCTOBER 15, 2017 AT 8:32 PM ~ EDIT

I agree with the place where the items have been placed. I also believe that a red colour scheme will be effective as it attracts the eye, thus will attract potential gamer's attention.

★ Like

Reply

Miron
OCTOBER 16, 2017 AT 7:24 PM ~ EDIT

I agree with the layout you have chosen. I think you should go for colours such as blue as it is bold but not so much that it will be unappealing

★ Like

Reply

walesowilsonsepq
OCTOBER 16, 2017 AT 9:47 PM ~ EDIT

Gushy, I think that this layout is fantastic! Perhaps you should pitch it to NinjaKiwi for the much anticipated Bloons TD7. Oli

★ Liked by 1 person

Reply

Nathan
OCTOBER 17, 2017 AT 8:36 AM ~ EDIT

I think that your layouts are good although I think that an enemy display isn't really necessary as the user could just use their own eyes to see what enemies are on the screen. Also, instead of a help button, there should be a tutorial instead. I think that there should be a light blue colour scheme as it looks natural and is easier to see compared to other colours such as red.

★ Like

Reply

For the implementation, I have decided to follow 10 Iterative Stages (which are explained in more detail in my Iterative Development Section). The stages are as follows:

- Iteration 1 - Basic Window, Sprite, Images and Click and Drag with money
- Iteration 2 – Enemies and Enemy Animation
- Iteration 3 – Weapons Animations
- Iteration 4 – Collisions
- Iteration 5 – Power Up (and Corresponding Collisions)
- Iteration 6 – Game Loop
- Iteration 7 – Speed
- Iteration 8 – Menus, Buttons and Help Screen
- Iteration 9 – High Scores
- Iteration 10 – Aesthetics and Sounds

See Appendix B for Test Plan During Development

The first test data to be used is to be used during the iterative development process to test the functionality of the game so far. The reasons for these tests for each iteration are as follows:

For the first iteration, simple test data is required:

- The opening of the application needs to be tested to ensure that the game can actually start up, as if it starts up here, it will start up in later stages too
- The click and drag function needs to be tested here. This is because the first iteration is largely being used to add weapons to the map using user mouse input and so at this stage, this needs to be confirmed to be working before proceeding to the next stages (to avoid encountering problems later on)

Iteration 2 Introduces enemies and so the test data is related to enemy addition which should be automatic upon pressing “run”:

- The enemies that are added should be the right amount according to the player’s score using the enemies algorithm described previously. Therefore, it is important to make sure this is working at the end of this iteration
- The enemies need to be added one after the other to ensure that they don’t all bunch together in one place and therefore make it hard for weapons to fire at the enemy (and also, if enemies are all bunched together, the game will be much harder for the user)
- The enemies need to follow the shape of the map as that is what the user expects them to do. If they aren’t doing this, then the game will not work properly and so it is important this is tested now.

Iteration 3 again should also run automatically:

- The weapons need to lock on to the enemy using the lock on algorithm described previously. If this is not done, then the game won't function
- The weapons also need to fire at the enemy given that it is within the correct distance
- The reset times are important as this is a major requirement of the problem specification and therefore needs to be tested at this stage

Iteration 4 involves adding collisions:

- When a weapon attacks an enemy, or an enemy attacks the main tower, it is important that the enemy or the tower's health goes down, and so this needs to be ensured for all enemies and towers and checked to be true at the end of this iteration
- If health of a tower or an enemy reaches zero, there needs to be some sort of feedback and removal of an object and this needs to be ensured by the code.

Iteration 5 – Powerup

- The power up is a key feature of the problem specification to be tested
- The power up needs to ensure that it can only be used once per mission. This needs to be tested to ensure it follows these rules. Equally, it needs to be tested to make sure that it is only bought given the user has enough money to buy it.
- In addition, the corresponding collisions need to be tested as well as this is another weapon

Iteration 6 – Game Loop

- As the simulation needs to run automatically without user input, it needs to be ensured that it is doing so and be tested at this stage

Iteration 7 – Speed

- This is part of the problem specification that needs to be checked to ensure that when the user clicks on a certain speed button, it increases/decreases the speed to the corresponding speed value and doesn't do something invalid

Iteration 8 – Menus, Help, Info

- All of these buttons and help screens need to be checked to ensure that they are working/functioning and are displaying the correct information. If they are not, then the user may not know what to do in a certain situation.

Iteration 9 – High Scores

- This is part of the problem specification and needs to be tested to ensure that the file is being accessed correctly and is displaying high scores in the correct order.
- It also needs to automate the high score display and add the high score to the correct place in the file by itself, and needs to be tested to ensure it does this

Iteration 10 – Aesthetics and sounds

- This is purely testing to ensure everything looks nice and sound effects that have been added play at the correct time. This is to ensure a seamless experience for the user

The testing during development is not limited to this, as this is only the black box test data and there will be a need for white box testing during development as well.

See Appendix C for Test Data for use After Development

This test data is largely testing against the problem specification. The purpose of this test data is for it to act as a form of acceptance testing, where each part of the problem specification is checked and if it has been fulfilled, it is ticked off.

Each of the test data refers to a certain aspect of the game. For example, the Menu section refers to all the parts of the problem specification that are to do with the menu. Each button on the menu needs to be tested to ensure it is doing what that button is supposed to do, such as “Start New Game” should start the game and allow the user to play, whereas the “Help” button should allow the user to gain access to any Help Information they require. This again is a form of black box testing.

Problem Being Tested	Action to be taken to test this problem	Expected Result	Actual Result
Iteration 1 - Basic Window, Sprite, Images and Click and Drag with money	Open Application	Application loads properly with all images in correct places as appropriate	
	Click and Drag a weapon to a correct part of the screen	Weapon is added to that part of the screen and correct money deducted if have enough	
	Click and Drag a weapon to an incorrect part of the screen	Error message is displayed	
Iteration 2 - Enemies and Enemy Animation	Click play button	Enemies added (not too many, not too little, must equal the current score of the player)	
	Click play button	Enemies added one after the other	
	Click play button	Enemies move one after the other following shape of map	
Iteration 3 - Weapons Animations	Click play button	Weapons follow enemy once the enemy is within their radii.	
	Click play button	Weapons fire at enemy when enemy is within their radii and have a reset time before firing again (make sure these times are followed)	
Iteration 4 - Collisions	Click play button	Weapons fire at enemy once the enemy is within their radii causing enemy health to decrease. If health of enemy is 0, enemy is removed from screen, score and money	
	Click play button	Enemy reaching end of path attack the tower causing tower health to decrease	
	Let enemies attack	If tower health is 0, game ends and message appears	
Iteration 5 - Powerup (and corresponding Collisions	Click and drag powerup during simulation if not already used once	Powerup added to part of screen and money is deducted from total money	
	Powerup attacks enemy	Enemy health decreases	
	Click and drag powerup during simulation if already used once	Error message is displayed	
	Click and drag normal weapon during simulation	Error message is displayed	
Iteration 6 - Game Loop	Zero enemies and tower still alive	Able to add more weapons again	
Iteration 7 - Speed	Click 0.5 speed button	Speed decreases	
	Click 2.0 speed button	Speed increases	
	Click 5.0 speed button	Speed increases	
Iteration 8 Menus and buttons and help screen	Help button pressed during simulation	Help screen appears	
	Automatic	Next wave of attackers appears on bottom	
	Pause button pressed	Gameplay is paused	
	User navigates away from Game Window	Gameplay is paused	
	Click on weapon and click sell weapon	Weapon is removed and money increased	
	Help button pressed from main menu	Help screen appears	
	Start New Game button pressed from main menu	Game begins	

Appendix B

Appendix B

Iteration 9 - Highscores	Highscores button pressed from main menu	Highscores shown	
	Tower health is zero	New highscore added and highscores shown before returning to main menu	
Iteration 10 - Aesthetics and sounds	Click play button	Sounds play at correct times	

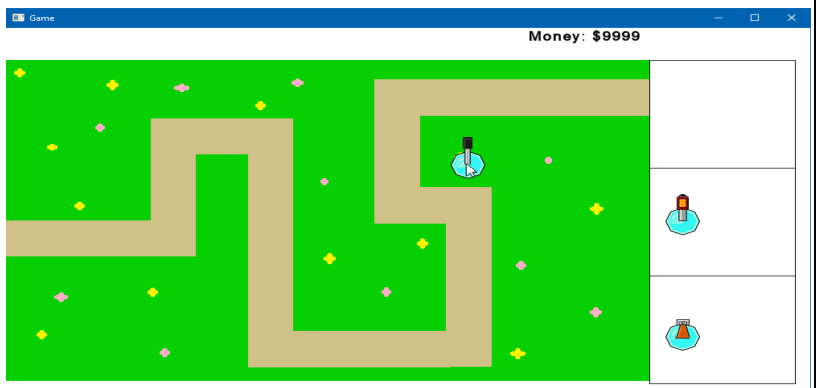
Appendix C

Problem Being Tested	Action to be taken to test this problem	Expected Result	Actual Result
Menu	Click on Help Button (See Help Button Testing Section Below)		
	Click on High Scores Button	Highscores should load up with scores in the correct place (judge by eye)	
	Click on New Game	New Game should be started from the	
Click and Drag Feature	Click on a weapon of choice and drag it to a valid part of the screen (i.e. to the game grid) with enough money	The weapon will be added to the position it is dragged to	
	Click on a weapon of choice and drag it to a valid part of the screen (i.e. to the game grid) without enough money	The weapon will not be added and an error message displaying "insufficient money" will be displayed (Or Words To That Effect)	
	Click on a weapon of choice and drag it to an invalid part of the screen (i.e. just outside of the game grid)	The weapon will not be added and an error message displaying "Weapon placed outside of game zone" will be displayed (OWTTE)	
	Click on a weapon of choice and drag it ontop of a previous weapon	The weapon will not be added and an error message displaying "Weapon placed in an invalid place" will be displayed (OWTTE)	
Play Button	Press the "Play" button to run the simulation	Enemies should be added to the screen and the game should bein to play itself	
Speed Buttons (these should be tested one after the other in the order given on the right)	Press the 0.5x speed button	Gameplay should slow down to approx. half the original speed (judge by eye)	
	Press the 1.0x speed button	Gameplay should return to it's original speed (judge by eye)	
	Press the 1.0x speed button	Gameplay should not change speed	
	Press the 2.0x speed button	Gameplay should speed up by approx. two times the original speed (judge by eye)	
	Press the 1.0x speed button	Gameplay should return to it's original speed (judge by eye)	
	Press the 5.0x speed button	Gameplay should speed up by approx. five times the original speed (judge by eye)	
Health bars	During the gameplay, one of the enemies should come under fire by a weapon	The enemy's health bar should decrease	
	During the gamepay, some enemies should reach the tower and attack the tower	The tower's health bar should decrease	
In Game Money System	Buy a weapon (using click and drag feature) with enough money	Total user money should decrease	
	Buy a weapon (using click and drag feature) without enough money	Error should be displayed saying "insufficient funds" (OWTTE)	
	Click on a weapon already placed in the game and click on "Sell Weapon"	Total user money should increase by amount the weapon returns	
	During the gameplay, one of the enemies should die	Total user money should increase by amount the enemy returns	
In Game Scoring System	During the gameplay, one of the enemies should die	Total user score should increase by amount the enemy returns	

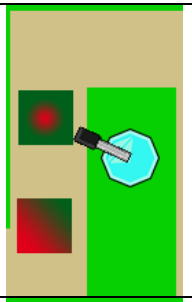
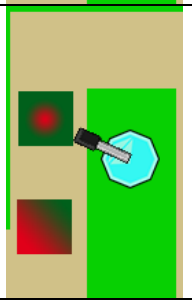
Appendix C

Emergency Powerups	During the gameplay, buy a powerup from the weapons menu using the click and drag feature (with enough money and not bought before)	The powerup should be placed where the user places it and the correct money deducted from the total user money	
	During the gameplay, buy a powerup from the weapons menu using the click and drag feature (with enough money but bought before)	An error message should appear stating "You have bought the powerup already. It cannot be used more than once" (OWTTE)	
	During the gameplay, buy a powerup from the weapons menu using the click and drag feature (without enough money and not bought before)	An error message should appear stating "You cannot afford the powerup" (OWTTE)	
	During the gameplay, buy a powerup from the weapons menu using the click and drag feature (without enough money and bought before)	An error message should appear stating "You cannot afford the powerup" (OWTTE)	
High scores	Should be automatic when the game ends	Highscores should load up with your score in the correct place (judge by eye) and with previous high scores	
Random Games	Play game 3 times	Each game should have different enemy combinations (write these combinations down in the box to the right)	
Game endless until failure	Play game 3 times	The game should not end until the health of the tower has reached zero each time	
Help Button	Click on Help Button	Screen should pop up showing help instructions to help you with playing the game	
	Click on close button on help screen	Screen should then disappear	
Information on next wave of enemies	Automatic	Should be displayed on bottom of screen	

Project Log

Date	Problem	Solution	Corresponding Tests
30/10/2017	The drag function drags from the corner of the weapon rather than from the centre	Subtract half of the Weapon's pixels to the mouse position so that it drags it from the middle	
31/10/2017	Lack of classes will cause major problems in the future	Classes Implemented and part of the code replaced with std::vector data structures to hold multiple objects of the same class	N/A
02/11/2017	Random function of enemies to add function is not random, but same each time	C++'s random function is not truly random. Instead, to make it slightly more random, I need to "seed" the random function with a seemingly random number. In my case, this can be the specific second at runtime.	N/A

Project Log

02/11/2017	Enemies move at different speeds for different frame rates	Speed = Distance/Time and so Speed*Time since last update should allow distance per second to be constant regardless of frame rate	Cannot be shown Tested and now works
02/11/2017	Enemies move at different speeds at different points	Make the float value for speed an integer instead as this gets the floor of the value and 1.5 is no longer considered greater than 1.01 etc.	Cannot Be Shown Tested and now works
05/11/2017	The weapon's point of rotation is not in the correct place	Set the point of rotation to halfway along the turret and at the base of the turret rather than the pre-set position	
05/11/2017	The weapon's turret is not following the enemy at all times	The angle of rotation is wrong and is different depending on where the enemy is. This is shown to be fixed in the Iteration 3 write up	
05/11/2017	The weapon often unlocks	Put the for loop for enemies outside of the weapon loop if	N/A

	when it should still be locked	the weapon has locked to ensure that the weapon does not unlock based on it not being locked on one enemy	
05/11/2017	Angle is in radians rather than degrees making it hard for it to follow at the correct angle	Multiply radian angle by (180/pi) to convert to degrees	N/A
12/11/2017	Game crashes when attempting to remove weapons at times	Do not delete the object before attempting to remove it from the vector	No longer crashes when run
15/11/2017	Sometimes, the weapons stop recognising enemies in its radius and does not lock onto them	LockToggle is true when it shouldn't be. It is better to simply remove it	Lock Toggle was useless so it now works perfectly
16/11/2017	The game lags out when enemies are moving too fast, causing the enemies to go off the path and never return to the path again	Have enemies be attracted to points on the map instead of checking if they are in a certain range	It no longer causes as many bugs when it lags and is much smoother overall
24/11/2017	Priority of the weapon is wrong. They lock onto the last	Put in break; once locked onto an enemy	Now locks onto the first enemy to enter

	enemy to enter the radius rather than the first		
24/11/2017	Game stutters at 0.25x speed	Remove this option	Now works
30/11/2017	Score saved in incorrect positions in file, especially when a score is repeated	Adapt the binary search algorithm to suit the program's specific needs	Now places in correct positions

Iterative Development Process

Ghusharib Chohan 605

My iterative stages have already been decided during the design stage and I will follow these stages. The benefits of these stages is that with every iteration, I will have a functioning program at the end of each one which can actually be played (albeit it may not be very fun). In addition, I have a running problem log which contains all the problems I come across during the development stage and how I have overcome these problems as problems are an inevitable part of this stage.

Iteration 1 – Basic Window, Sprite, Images and Click & Drag Weapons Modules

In this first iteration, the general window and setup of the game is created in terms of window size, the map and most importantly the click & drag weapon function to allow the user to add weapons to the map. This is the most basic iteration and contains some of the most basic but vital parts of the game and therefore had to be part of the first iteration.

The first important part was creating a new window with the right size, title and the sprites that would be visible on the screen. To do this, I had the following code:

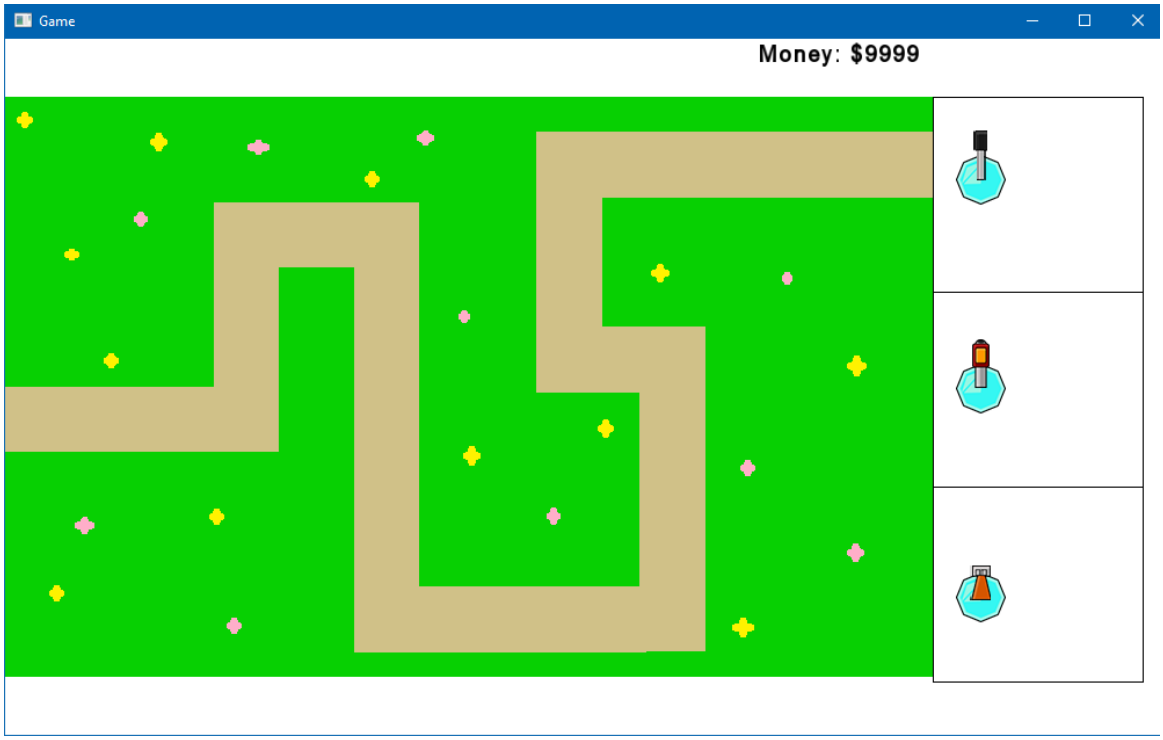
```
#include <SFML/Graphics.hpp>
#include <string>
#include <sstream>
#include <iostream>

class Game
{
  public:
      Game()
          : mWindow(sf::VideoMode(1000, 600), "Game"), Weapon1BaseSprite(), Weapon1BaseTexture(), Weapon1TurretSprite(), Weapon1TurretTexture(),
            Weapon1Border(), Weapon2BaseSprite(), Weapon2BaseTexture(), Weapon2TurretSprite(), Weapon2TurretTexture(), Weapon2Border(),
            Weapon3BaseSprite(), Weapon3BaseTexture(), Weapon3TurretSprite(), Weapon3TurretTexture(), Weapon3Border(), mapBackground(), Radius(),
            Arial()

          {
            //In the above code, we are handling all the possible variables such as Weapon1Base, Weapon1Turret etc. and setting their values
            //Whenever a new object of the class "Game" is made
            //mWindow relates to the game window which opens and is set to 1000pixels by 600pixels and has a title "Game"

            //Background
            if (!bTexture.loadFromFile("Images/Background.jpg"))
            {
                //Handle Loading Error
            }
            mBackground.setTexture(bTexture);
            //The image Background.jpg contains a simple white rectangle image. The code sets the background using the texture from this image

            if (!mapBackgroundTexture.loadFromFile("Images/Map.png"))
            {
                //Handle Loading Error
            }
            mapBackground.setTexture(mapBackgroundTexture);
            //The map is added on to the screen using a sprite and assigning a texture to the sprite
            mapBackground.setPosition(0, 50);
            //The position of the map is set

            //Weapon1
            if (!Weapon1BaseTexture.loadFromFile("Images/MainBase.png"))
            {
                //Handle Loading Error
            }
            Weapon1BaseSprite.setTexture(Weapon1BaseTexture);
            if (!Weapon1TurretTexture.loadFromFile("Images/Weapon1Top.png"))
            {
                //Handle Loading Error
            }
            Weapon1TurretSprite.setTexture(Weapon1TurretTexture);

            Weapon1BaseSprite.setPosition(820.f, 100.f);
            Weapon1TurretSprite.setPosition(Weapon1BaseSprite.getPosition().x + 15.5f, Weapon1BaseSprite.getPosition().y - 21.25f);

            Weapon1Border.setSize(sf::Vector2f(180, 167));
            Weapon1Border.setOutlineColor(sf::Color::Black);
            Weapon1Border.setOutlineThickness(1.f);
            Weapon1Border.setPosition(801.f, 51.f);

            //Weapon1 and it's corresponding sprites are loaded and placed onto the screen
            //Weapon1's border is also given values so that a border appears around the weapon as according to the problem specification
```

The above code corresponds to what should automatically happen every time I create a new object of the “Game” class. Whenever a new object is made, the variables of the class which need to be pre-defined are shown. The code creates a window of size 1000pixels wide by 600pixels high and adds to it a white background, the map, the weapons that are able to be added and the weapon borders to be in line with the user interfaces shown in the problem specification. The next step is to draw on the sprites/backgrounds onto the actual window and this is done by using

```
mWindow.clear();
mWindow.draw(mBackground);
mWindow.draw(mapBackground);
mWindow.draw(Weapon1Border);
mWindow.draw(Weapon2Border);
mWindow.draw(Weapon3Border);
mWindow.draw(Weapon1BaseSprite);
mWindow.draw(Weapon1TurretSprite);
mWindow.draw(Weapon2BaseSprite);
mWindow.draw(Weapon2TurretSprite);
mWindow.draw(Weapon3BaseSprite);
mWindow.draw(Weapon3TurretSprite);
//mWindow.draw() draws on all the sprites that need to be drawn to create a screen with visible images
```

the Window.draw() function as shown below:

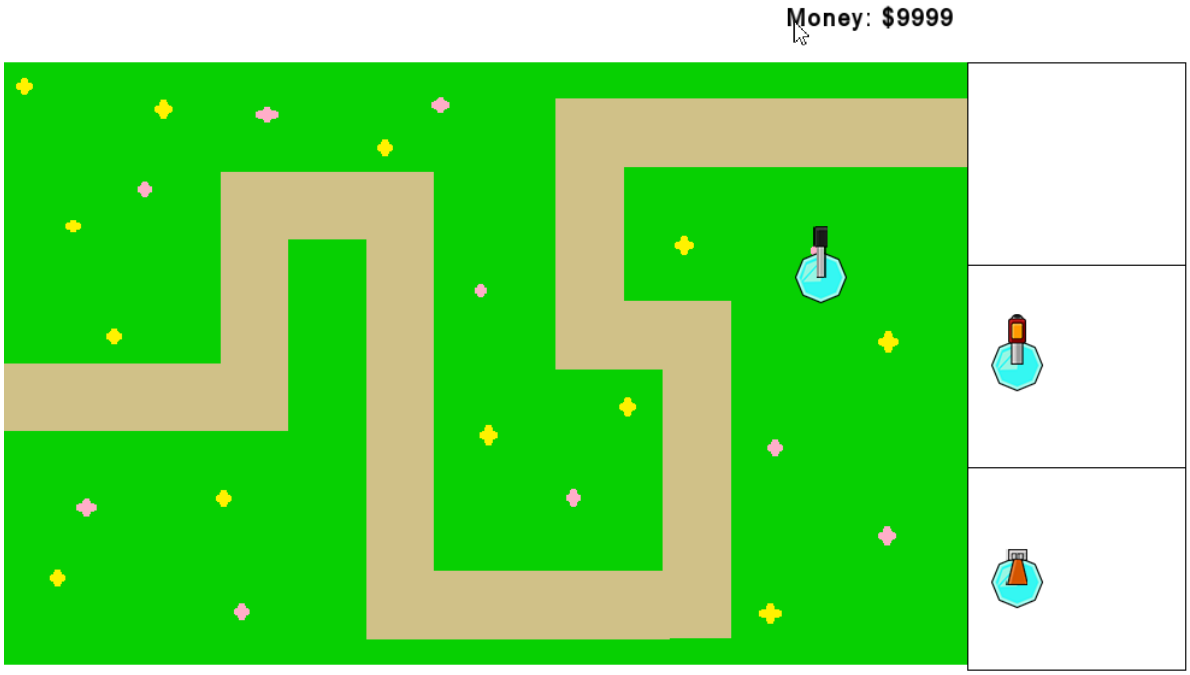
This successfully managed to get me the right initial screen display:



For the click and drag function, I had to create some functions within my game class. I used the SFML library's event variable to check for mouse movements and actions such as Mouse Left Button Press, Mouse Moving and the Mouse being released. I could also get the position of these values which helped me gauge where the user's mouse was at a certain point in time and so therefore I would be able to check if a user was clicking on a Weapon or not (and so therefore work out whether or not I needed to consider dragging that weapon). For example, for my mouseButtonPressedEvent, I had the following

```
184                 case sf::Event::MouseButtonPressed:
185                     if(event.mouseButton.button == sf::Mouse::Left)
186                     {
187                         if (event.mouseButton.x > Weapon1BaseSprite.getPosition().x &&
188                             event.mouseButton.x < Weapon1BaseSprite.getPosition().x + 42.5f &&
189                             event.mouseButton.y > Weapon1BaseSprite.getPosition().y &&
190                             event.mouseButton.y < Weapon1TurretSprite.getPosition().y + 64.25f)
191                             //If the mouse is pressed anywhere between the range of the Weapon1Base's top left corner, top right corner
192                             //bottom left corner and bottom right corner, then we know that it is being pressed on
193                         {
194                             isDragging1 = true;
195                             //And therefore we can safely assume that this Weapon is about to be dragged onto the screen
196                         }
197                         else if (event.mouseButton.x > Weapon2BaseSprite.getPosition().x &&
198                                  event.mouseButton.x < Weapon2BaseSprite.getPosition().x + 42.5f &&
199                                  event.mouseButton.y > Weapon2BaseSprite.getPosition().y &&
200                                  event.mouseButton.y < Weapon2TurretSprite.getPosition().y + 63.25f)
201                                  //Same as above
202                         {
203                             isDragging2 = true;
204                         }
205                         else if (event.mouseButton.x > Weapon3BaseSprite.getPosition().x &&
206                                  event.mouseButton.x < Weapon3BaseSprite.getPosition().x + 42.5f &&
207                                  event.mouseButton.y > Weapon3BaseSprite.getPosition().y &&
208                                  event.mouseButton.y < Weapon3TurretSprite.getPosition().y + 51.25)
209                         {
210                             isDragging3 = true;
211                         }
212                     }
213                     break;
```

The above code checks the four corners of the weapon sprite and if the user clicks anywhere within that area, then the weapon can be dragged. The code (not shown) later goes on to make the weapon which is being dragged follow the mouse cursor to where it goes while the button is being pressed and adds in the weapon only once it has been released.



However, as can be seen in my problem log and in the photo above, a problem came about when attempting to drag the weapon as instead of the weapon being dragged from the middle (as would be aesthetically pleasing) it dragged from the top left corner. The weapon is being dragged on but the cursor is not in the middle.

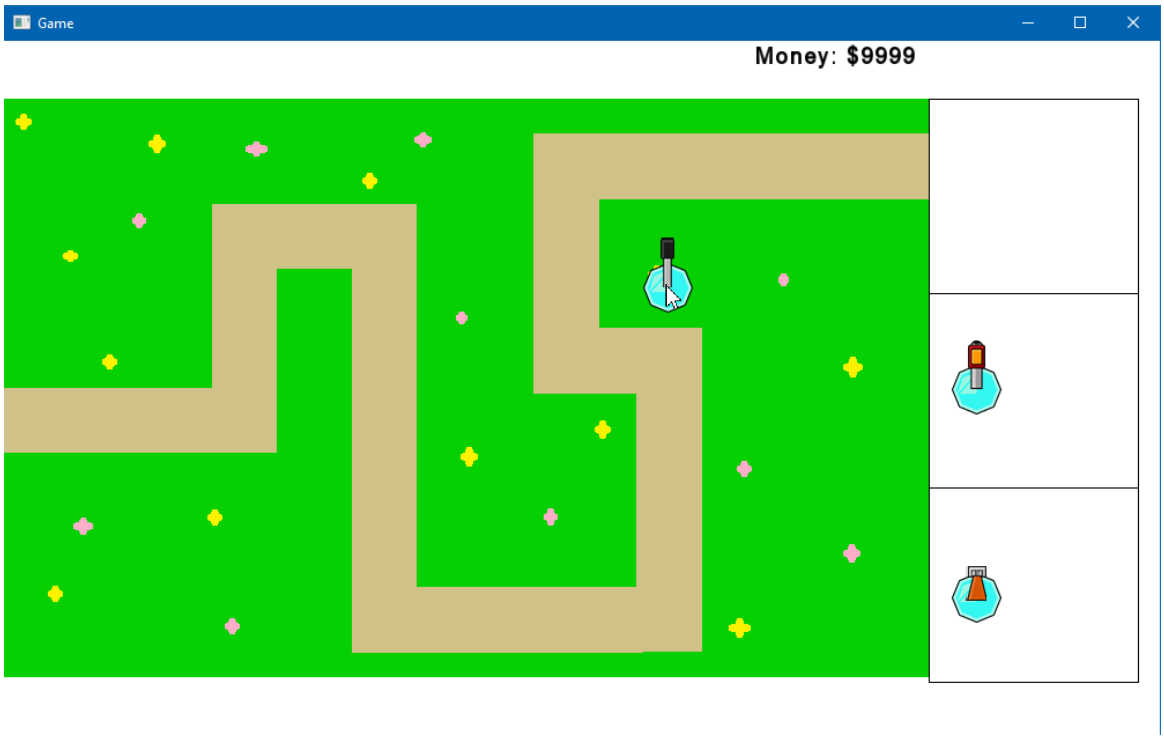
The reason for this lies in the code for the event when the mouse is being moved:

```
if (isDragging1)
{
    Weapon1BaseSprite.setPosition(event.mouseMove.x, event.mouseMove.y);
    Weapon1TurretSprite.setPosition(Weapon1BaseSprite.getPosition().x + 15.5f, Weapon1BaseSprite.getPosition().y - 21.25f);
}
```

What this does is that it places the Weapon1BaseSprite in the same place as the cursor, but the origin of the Sprite is in the top left corner. This means that the code fails to recognise that the Weapon needs to be placed using the middle of the sprite rather than the corner, but there are two simple ways of fixing this. The first method is to change the origin using the MySprite.setOrigin(int, int) function. However, this can affect later transformations so it is easier to simply subtract half of the Weapon1's pixels to the position as this would work too as shown below:

```
if (isDragging1)
{
    Weapon1BaseSprite.setPosition(event.mouseMove.x - 21.25, event.mouseMove.y - 21.25);
    Weapon1TurretSprite.setPosition(Weapon1BaseSprite.getPosition().x + 15.5f, Weapon1BaseSprite.getPosition().y - 21.25f);
}
```

And now it works:



The next problem was not a problem stopping my code from working, but would make the program much harder to create. This comes from the fact that my Weapons are not using classes but instead creating new sprites for every value. This would create a problem as I've already defined classes in my Design Section and as there are going to be three kinds of weapons with different variables, it is important to have a state of consistency to enable me to make my code easier in the later stages. This was overcome by implementing these classes as follows:

```
55  class Weapon1Class
56  {
57      public:
58      Weapon1Class()
59          : BaseSprite(), BaseTexture(), TurretSprite(), TurretTexture()
60          {
61              if(!BaseTexture.loadFromFile("Images/MainBase.png"))
62              {
63                  //Handle Loading Error
64              }
65              BaseSprite.setTexture(BaseTexture);
66
67              if(!TurretTexture.loadFromFile("Images/Weapon1Top.png"))
68              {
69                  //Handle Loading Error
70              }
71              TurretSprite.setTexture(TurretTexture);
72
73              BaseSprite.setPosition(820.f, 100.f);
74              TurretSprite.setPosition(BaseSprite.getPosition().x + 15.5f, BaseSprite.getPosition().y - 21.25f);
75
76              Border.setSize(sf::Vector2f(180, 167));
77              Border.setOutlineColor(sf::Color::Black);
78              Border.setOutlineThickness(1.f);
79              Border.setPosition(801.f, 51.f);
80
81          }
82
83      public:
84          sf::Vector2f getBasePosition()
85          {
86              return BaseSprite.getPosition();
87          }
88          sf::Vector2f getTurretPosition()
89          {
90              return TurretSprite.getPosition();
91          }
92          void setBasePosition(float x, float y)
93          {
94              BaseSprite.setPosition(x, y);
95          }
96          void setTurretPosition(float x, float y)
97          {
98              TurretSprite.setPosition(x, y);
99          }
100         sf::Sprite getBaseSprite()
101         {
102             return BaseSprite;
103         }
104         sf::Sprite getTurretSprite()
105         {
106             return TurretSprite;
107         }
108         sf::RectangleShape getBorder()
109         {
110             return Border;
111         }
112         sf::Texture BaseTexture, TurretTexture;
113     private:
114         sf::Sprite BaseSprite, TurretSprite;
115         sf::RectangleShape Border;
116 };
```

The above class has sprite classes for the base and the turret as was in the previous code as well as a Rectangle Shape to store the border which was also part of the previous code. It also involves getter and setter methods for getting the sprites, the borders and positions as well as setting the positions. Whenever an object is created of this class, some variables are pre-defined for it such as the Weapon Position, the Weapon Texture and so on.

```
8       public:
9           GeneralWeaponClass():
10              BaseSprite(), BaseTexture()
11          {
12              if(!BaseTexture.loadFromFile("Images/MainBase.png"))
13              {
14                  //HandleLoadingError
15              }
16              BaseSprite.setTexture(BaseTexture);
17          }
18      public:
19          sf::Vector2f getBasePosition()
20          {
21              return BaseSprite.getPosition();
22          }
23          sf::Vector2f getTurretPosition()
24          {
25              return TurretSprite.getPosition();
26          }
27          void setBasePosition(float x, float y)
28          {
29              BaseSprite.setPosition(x, y);
30          }
31          void setTurretPosition(float x, float y)
32          {
33              TurretSprite.setPosition(x, y);
34          }
35          sf::Sprite getBaseSprite()
36          {
37              BaseSprite.setTexture(BaseTexture);
38              return BaseSprite;
39          }
40          sf::Sprite getTurretSprite()
41          {
42              TurretSprite.setTexture(TurretTexture);
43              return TurretSprite;
44          }
45          void setTurretTexture(sf::Texture turret)
46          {
47              TurretTexture = turret;
48              TurretSprite.setTexture(turret);
49          }
50          sf::Texture BaseTexture, TurretTexture;
51      private:
52          sf::Sprite BaseSprite, TurretSprite;
53  };
```

A general weapon class was also created which contains pretty much the same variables but with some differences e.g. it has no border. This has the benefit that once the user drags a Weapon of any of the 3 types onto the map, I can then make it an object of the general weapon class and so all the weapons are part of the same class type. I can then make a vector of class GeneralWeaponClass and add all the weapons to this vector instead of having individual sprites for the numerous weapons a user may enter.

```
class Game
  {
    public:
        Game()
            : mWindow(sf::VideoMode(1000, 600), "Game"), Weapon1(), Weapon2(), Weapon3(), mapBackground(), Radius(), Arial()

            {
```

```
507     private:
508         Weapon1Class Weapon1;
509         Weapon2Class Weapon2;
510         Weapon3Class Weapon3;
511
512         //Window
513         sf::RenderWindow mWindow;
514
515         //Sprites
516         sf::Sprite mBackground, mapBackground;
517         std::vector<GeneralWeaponClass> Weapons;
518
519         //Textures
520         sf::Texture bTexture, mapBackgroundTexture;
521
522         //Shapes
523         sf::CircleShape Radius;
524
525         //Fonts
526         sf::Font Arial;
527
528         //Text
529         sf::Text MoneyText;
530
531         //Variables
532         std::vector<int> WeaponTypes;
533         int Money = 99999;
534         int totalWeapons = 0;
535         float PlayerSpeed = 100;
536         bool isDragging1 = false, isDragging2 = false, isDragging3 = false, showCross = false, toggle = false;;
```

By creating objects of the individual weapon type classes, I could then cut down on the vast number of variables I had previously had in my game class simplifying the code

A vector of type GeneralWeaponClass to store all the user's Weapons in one single accessible data structure

This also simplified my addNewWeapon function as it could now simply create a generalWeapon vector instead of a separate vector for turrets and bases (and this old method would have led me to having to create further vectors for damages, textures and everything in between so classes are clearly better).

Before:

```
291         void addNewWeapon(sf::Vector2f position, sf::Vector2f turret, int WeaponType)
292         {
293             if (position.x < 100 && position.y > 500)
294             {
295                 //Do not do anything
296             }
297             else
298             {
299                 sf::Sprite newWeaponBase;
300                 newWeaponBase.setTexture(Weapon1BaseTexture);
301                 newWeaponBase.setPosition(position);
302                 sf::Sprite newWeaponTurret;
303                 switch (WeaponType){
304                     case 1:
305                     newWeaponTurret.setTexture(Weapon1TurretTexture);
306                     Money -= 2500;
307                     break;
308 
309                     case 2:
310                     newWeaponTurret.setTexture(Weapon2TurretTexture);
311                     Money -= 5000;
312                     break;
313 
314                     case 3:
315                     newWeaponTurret.setTexture(Weapon3TurretTexture);
316                     Money-= 10000;
317                     break;
318                 }
319                 newWeaponTurret.setPosition(turret);
320                 WeaponBases.push_back(newWeaponBase);
321                 WeaponTurrets.push_back(newWeaponTurret);
322                 WeaponTypes.push_back(WeaponType);
323                 totalWeapons++;
324             }
325         }
```

After:

```
430         void addNewWeapon(sf::Vector2f position, sf::Vector2f turret, int WeaponType)
431         {
432             if (position.x < 100 && position.y > 500)
433             {
434                 //Do not do anything
435             }
436             else
437             {
438                 GeneralWeaponClass newWeapon;
439                 newWeapon.setBasePosition(position.x, position.y);
440                 switch (WeaponType){
441                     case 1:
442                     newWeapon.setTurretTexture(Weapon1.TurretTexture);
443                     Money -= 2500;
444                     break;
445 
446                     case 2:
447                     newWeapon.setTurretTexture(Weapon2.TurretTexture);
448                     Money -= 5000;
449                     break;
450 
451                     case 3:
452                     newWeapon.setTurretTexture(Weapon3.TurretTexture);
453                     Money-= 10000;
454                     break;
455                 }
456                 newWeapon.setTurretPosition(turret.x, turret.y);
457                 Weapons.push_back(newWeapon);
458                 totalWeapons++;
459             }
460         }
```

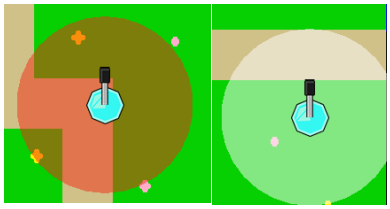
In the “after” code I only need to add a weapon object of the generalWeaponClass to a single vector.

A hard point of the click and drag function was realising when the mouse was moved above a point on the map at which the weapon could not be added. This meant checking multiple pixels and made for very ugly code, but there was no better method. Therefore, my mouse moved event code ended up looking like this:

```
99                  case sf::Event::MouseMoved:
100                     for (int i = 0; i < totalWeapons; i++)
101                     {
102                         if (event.mouseMove.x < Weapons[i].getBasePosition().x + 63.75 &&
103                             event.mouseMove.x > Weapons[i].getBasePosition().x  - 21.25 &&
104                             event.mouseMove.y < Weapons[i].getBasePosition().y + 85 &&
105                             event.mouseMove.y > Weapons[i].getBasePosition().y - 42.5)
106                         {
107                             Radius.setFillColor(sf::Color(242,43,21,128));
108                             toggle = true;
109                             break;
110                         }
111                     }
112                     if (event.mouseMove.x > 0 && event.mouseMove.x < 257.25 && event.mouseMove.y > 267.25 && event.mouseMove.y < 418.75)
113                         Radius.setFillColor(sf::Color(242,43,21,128));
114                     else if (event.mouseMove.x < 257.25 && event.mouseMove.x > 136.5 && event.mouseMove.y < 256.5 && event.mouseMove.y > 110)
115                         Radius.setFillColor(sf::Color(242,43,21,128));
116                     else if (event.mouseMove.x < 385.25 && event.mouseMove.x > 136.5 && event.mouseMove.y < 238.5 && event.mouseMove.y > 110)
117                         Radius.setFillColor(sf::Color(242,43,21,128));
118                     else if (event.mouseMove.x < 385.25 && event.mouseMove.x > 278.25 && event.mouseMove.y > 110 && event.mouseMove.y < 550)
119                         Radius.setFillColor(sf::Color(242,43,21,128));
120                     else if (event.mouseMove.x < 625.25 && event.mouseMove.x > 278.25 && event.mouseMove.y > 449.75 && event.mouseMove.y < 550)
121                         Radius.setFillColor(sf::Color(242,43,21,128));
122                     else if (event.mouseMove.x < 625.25 && event.mouseMove.x > 524.25 && event.mouseMove.y > 225.75 && event.mouseMove.y < 550)
123                         Radius.setFillColor(sf::Color(242,43,21,128));
124                     else if (event.mouseMove.x < 625.25 && event.mouseMove.x > 435.75 && event.mouseMove.y > 225.75 && event.mouseMove.y < 325.25)
125                         Radius.setFillColor(sf::Color(242,43,21,128));
126                     else if (event.mouseMove.x < 536.25 && event.mouseMove.x > 435.75 && event.mouseMove.y < 283.75)
127                         Radius.setFillColor(sf::Color(242,43,21,128));
128                     else if (event.mouseMove.x > 515 && event.mouseMove.y < 157.25)
129                         Radius.setFillColor(sf::Color(242,43,21,128));
130                     else if (event.mouseMove.x < 25 || event.mouseMove.x > 750 || event.mouseMove.y < 75 || event.mouseMove.y > 525)
132                     else if(!toggle)
133                         Radius.setFillColor(sf::Color(255,255,255,128));
134
135
136                     if (isDragging1)
137                     {
138                         Weapon1.setBasePosition(event.mouseMove.x - 21.25f, event.mouseMove.y - 21.25f);
139                         Weapon1.setTurretPosition(Weapon1.getBasePosition().x + 15.5f, Weapon1.getBasePosition().y - 21.25f);
140                     }
141                     if (isDragging2)
142                     {
143                         Weapon2.setBasePosition(event.mouseMove.x - 21.25, event.mouseMove.y - 21.25);
144                         Weapon2.setTurretPosition(Weapon2.getBasePosition().x + 14.f, Weapon2.getBasePosition().y - 21.25f);
145                     }
146                     if (isDragging3)
147                     {
148                         Weapon3.setBasePosition(event.mouseMove.x - 21.25, event.mouseMove.y - 21.25);
149                         Weapon3.setTurretPosition(Weapon3.getBasePosition().x + 12.5f, Weapon3.getBasePosition().y - 6.25f);
150                     }
151                     Radius.setPosition(event.mouseMove.x - 100, event.mouseMove.y - 100);
152                     break;
```

And so by checking where the mouse button was released, I could then simply create a new object of the general weapon class with its position in the very place where the mouse had been. This worked perfectly.

It should also be noted that in the above code, I change the radius colour changes if the weapon is hovering over an invalid place. This was to achieve part of the problem specification and to have feedback to the user of when the weapon was being added to an incorrect place (and makes it easy to understand):

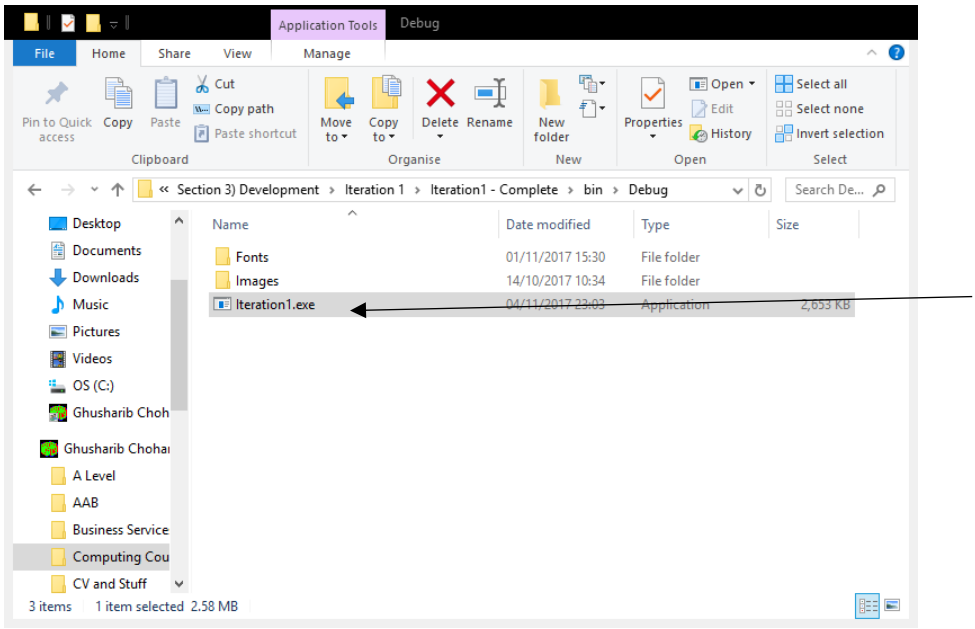


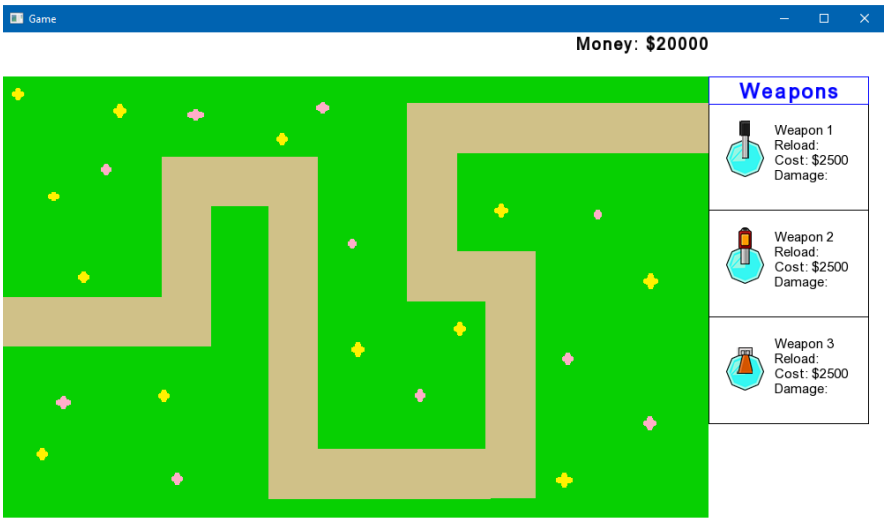
Attempting to add a weapon to an invalid place vs a valid place. This radius also shows the range of a weapon instantly to the user as per the requirements.

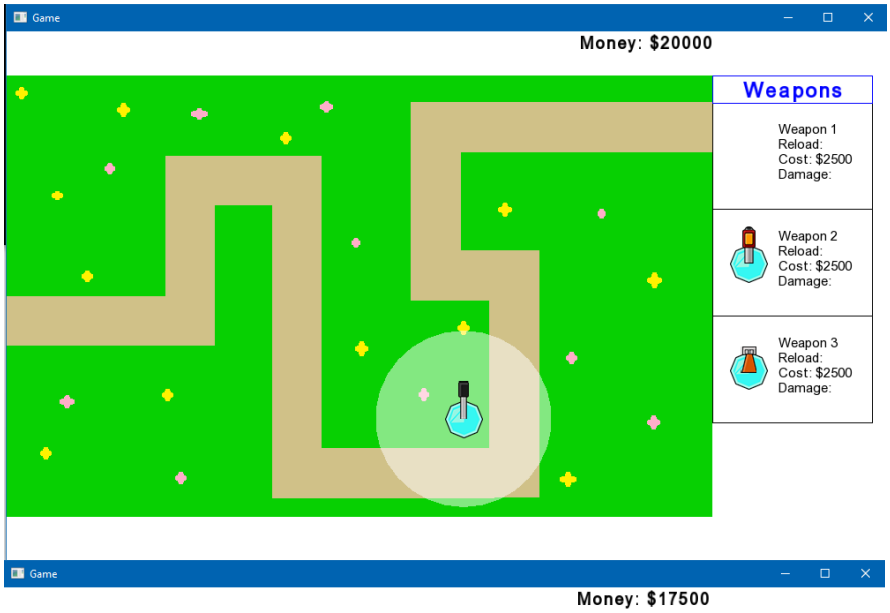
Tests for Iteration 1:

	Problem Being Tested	Action to be taken to test this problem	Expected Result
2	Iteration 1 - Basic Window, Sprite, Images and Click and Drag with money	Open Application	Application loads properly with all images in correct places as appropriate
3		Click and Drag a weapon to a correct part of the screen	Weapon is added to that part of the screen and correct money deducted if have enough money
4		Click and Drag a weapon to an incorrect part of the screen	Error message is displayed

Opening the application:

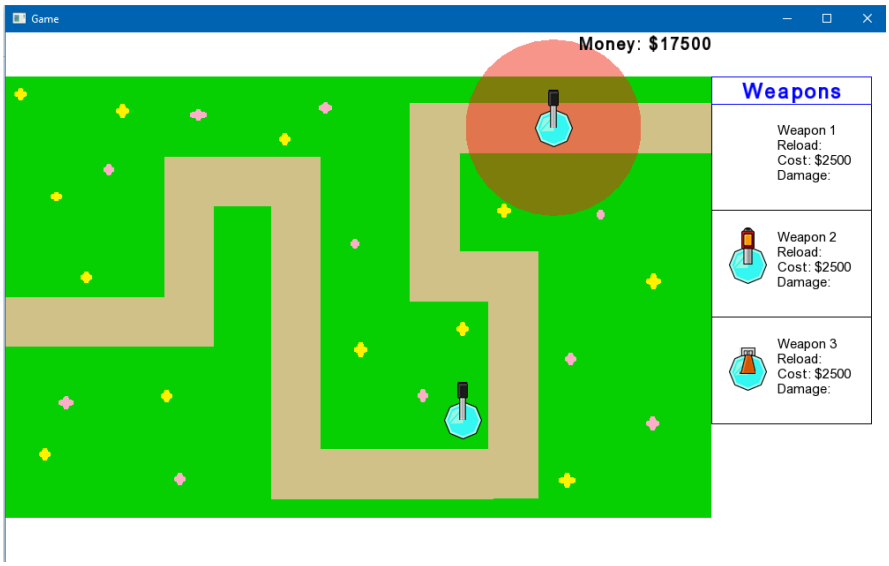




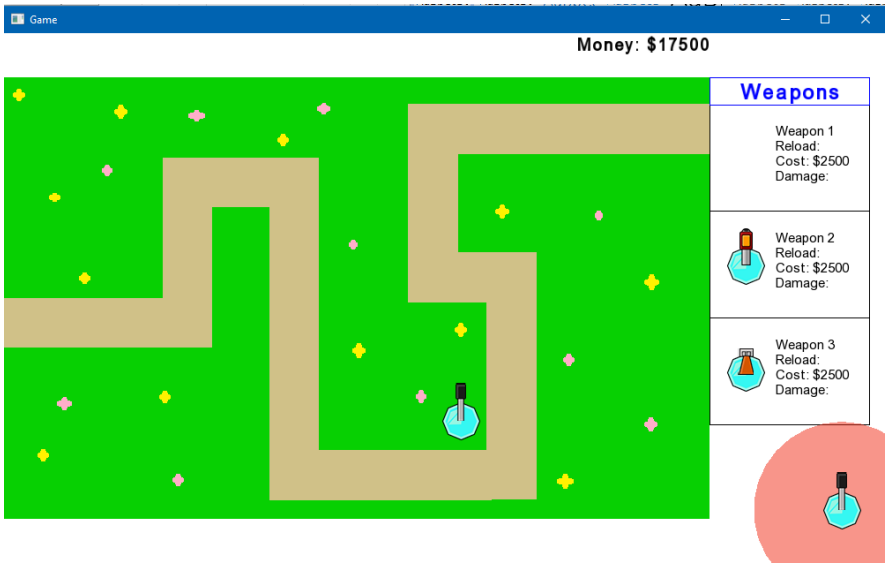


Test 2) A weapon is dragged to a valid point on the map as shown by the white radius

The money decreases as per the test requirement



Test 3) Weapon is dragged to an invalid part of the screen. It is not added and feedback comes from the red radius



A further test for test 3

Therefore, Iteration 1 is complete with a functioning game whereby you are able to add weapons to a map (as many as you like) and all the black box tests have been passed by it! I can now progress to my second iteration.

Iteration 2 – Enemy Classes and Enemy Movement Modules

In this iteration, there is nothing difficult as such to add. All that needs to be done is for the type of enemy to be added to be decided by the algorithm defined back in the design stage and for the enemies to follow the path of the map. However, it is an important part for the functionality of the game.

My “enemiesToAdd()” function follows the pseudocode as defined back in the design stage of my game. In the beginning, the function was not random as the C++ rand() function is not truly random, but by seeding the random function with the specific point in time the game was being run, I was able to overcome this challenge (as is shown in the below code). This was to ensure each game was random and is as follows:

```
138         void typeOfEnemyToAdd()
139         {
140             int TempScore = Score;
141             srand((int)time(0));
142             //This seeds the random function using the time the game is being run at
143             while (TempScore > 0)
144             {
145                 int n = (rand() % 3) + 1; //This get's a random integer being either 1, 2 or 3
146                 switch (n)
147                 {
148                 case 1:
149                     addEnemy(1);
150                     TempScore = TempScore - 10;
151                     break;
152                     //If the random number is 1, then the first kind of enemy is added into the game
153                     //This enemy can always be added, and so I do not need to check if the score allows me to add this enemy or not
154 
155                 case 2:
156                     if (TempScore - 50 >= 0)
157                     {
158                         addEnemy(2);
159                         TempScore = TempScore - 50;
160                     }
161                     break;
162                     //If the random number is 2, then the second kind of enemy is added into the game
163                     //However, I need to check this can be added before adding it as if the score falls below 50, this means that the
164                     //user has not progressed far enough in the game to be able to combat this enemy
165 
166                 case 3:
167                     if (TempScore - 100 >= 0)
168                     {
169                         addEnemy(3);
170                         TempScore = TempScore - 100;
171                     }
172                     break;
173                     //If the random number is 3, then the third kind of enemy is added into the game
174                     //However, I need to check this can be added before adding it as if the score falls below 100, this means that the
175                     //user has not progressed far enough in the game to be able to combat this enemy
176                 }
177             }
178             enemiesAdded = true;
179             //I then need to set my toggle to true to tell the game that enemies have now been added and that we can progress with the game
180         }
```

The code is exactly the same as my predefined pseudocode. The code also checks to make sure the user has progressed far enough in the game before adding the tougher enemies, and provides a proper random game each time.

The addEnemies() function that follows each enemy addition creates a new object of the generalEnemyClass() and adds to the Enemies[] vector much like the addWeapon algorithm back in Iteration1 did (but obviously for enemies instead):

```
182         void addEnemy(int type)
183         {
184             GeneralEnemyClass newEnemy;
185             switch (type)
186             {
187                 case 1:
188                     newEnemy.setTexture(Enemy1.texture);
189                     newEnemy.setScore(Enemy1.getScore());
190                     break;
191                 case 2:
192                     newEnemy.setTexture(Enemy2.texture);
193                     newEnemy.setScore(Enemy2.getScore());
194                     break;
195                 case 3:
196                     newEnemy.setTexture(Enemy3.texture);
197                     newEnemy.setScore(Enemy3.getScore());
198                     break;
199                 //Depending on what number enemy is being added, the relevant texture and score is given to the object
200             }
201 
202             if (totalEnemies == 0)
203                 {
204                     newEnemy.setPosition(0, 310);
205                     //The first enemy has a constant position
206                 }
207             else
208                 {
209                     newEnemy.setPosition(Enemies[totalEnemies - 1].getPosition().x - 100, Enemies[totalEnemies - 1].getPosition().y);
210                     //All enemies after it are placed 100pixels after the other
211                 }
212             Enemies.push_back(newEnemy);
213             totalEnemies++;
214             //The enemy is pushed onto the vector and the totalEnemies count is updated
215         }
```

The algorithm simply takes in which enemy is being added and adds it to the enemies vector to be later rendered onto the screen (and updates the total enemies count)

```
226         void render()
227             {
228                 mWindow.clear();
229                 mWindow.draw(mBackground);
230                 mWindow.draw(mapBackground);
231                 mWindow.draw(Weapon1.getBorder());
232                 mWindow.draw(Weapon2.getBorder());
233                 mWindow.draw(Weapon2.getBorder());
234                 mWindow.draw(Weapon3.getBorder());
235                 if (isDragging1 || isDragging2 || isDragging3)
236                 {
237                     mWindow.draw(Radius);
238                 }
239                 mWindow.draw(Weapon1.getBaseSprite());
240                 mWindow.draw(Weapon1.getTurretSprite());
241                 mWindow.draw(Weapon2.getBaseSprite());
242                 mWindow.draw(Weapon2.getTurretSprite());
243                 mWindow.draw(Weapon3.getBaseSprite());
244                 mWindow.draw(Weapon3.getTurretSprite());
245 
246 
247                 for(int i = 0; i < totalWeapons; i++)
248                 {
249                     mWindow.draw(Weapons[i].getBaseSprite());
250                     mWindow.draw(Weapons[i].getTurretSprite());
251                 }
252                 std::stringstream s;
253                 s << Money;
254                 MoneyText.setString("Money: $" + s.str());
255                 mWindow.draw(MoneyText);
256                 mWindow.draw(Weapon1Text);
257                 mWindow.draw(Weapon2Text);
258                 mWindow.draw(Weapon3Text);
259                 mWindow.draw(WeaponsTitleBorder);
260                 mWindow.draw(WeaponsTitle);
261                 if(!playingGame)
262                 {
263                     mWindow.draw(PlayButtonBorder);
264                     mWindow.draw(PlayButton);
265                 }
266                 else
267                 {
268                     for (int i = 0; i < totalEnemies; i++)
269                     {
270                         mWindow.draw(Enemies[i].getEnemySprite());
271                     }
272                 }
273                 mWindow.display();
274                 toggle = false;
275             }
```

The code on the left shows the render function, which simply renders all the sprites. Note the use of for loops to render all the weapons in the Weapons[] vector and all the enemies in the Enemies vector

To keep my enemies moving the same distance per second (in essence speed) regardless of the frame rate of the machine it is being run on, I introduced a clock to the game using the SFML sf::clock class.

The way this works comes down to the classic physics equation $speed = \frac{distance}{time}$. In my case, speed is constant, but depending on the framerate, the distance travelled per second changes This means that on a PC with a low frame rate, the distance travelled in a second may be lower than on a PC with a high frame rate, which is bad and we want it to be kept constant. If we get the time per frame, we can multiply the speed by the time per frame and get a constant distance per second. For example:

Suppose a PC runs at 30FPS and another at 60FPS. Originally, the 30FPS machine would travel 60pixels on a 2pixels per second constant speed, and 120pixels on the 60FPS machine.

However, by using the equation above, we can make the distance travelled constant. This is because the time per frame for a 30FPS machine is 1/30 seconds and 1/60 seconds for a 60FPS Machine. Therefore, 2 * 1/30 makes 2/30 pixels and 1/60 * 2 makes 2/60 pixels. So the 30FPS machine will travel 2/30 pixels every 1/30 seconds making 2 pixels per second and the 60FPS machine will travel 2/60 pixels every 1/60 seconds also making 2 pixels per second.

The second problem came from the enemy travelling at different speeds along the path. This came from the fact that my speed value was a float value. This meant that at some points, the speed of the enemy as 1.2 and at others 1.5 causing a visible increase/decrease in speed. By making the enemy speed an integer, I was able to make the speed of the enemy be equal to the floor of the float speed. In other words, both 1.2 and 1.5 were treated as 1 rather than two different speeds, allowing for constant speed throughout the path:

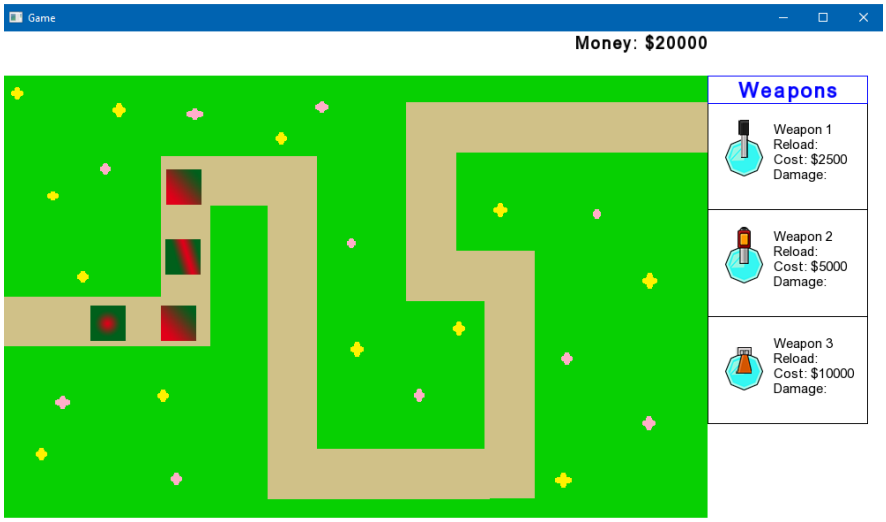
```
90          void run()
91          {
92              sf::Clock clock;
93              while (mWindow.isOpen())
94              {
95                  if(!playingGame)
96                  {
97                      processPreGameEvents();
98                      clock.restart();
99                      //If weapons are still being added, pre game events will be processed
100                 }
101                 else
102                 {
103                     processDuringGameEvents();
104                     if (clock.getElapsedTime().asSeconds() > TimePerFrame)
105                     {
106                         update(clock.getElapsedTime());
107                         //The enemies movement will be updated with the time elapsed since the last update sent to the update function
108                         clock.restart();
109                         //The clock is then restarted to calculate the time for the next frame
110                     }
111                 }
112                 render();
113                 //The window is rendered
114             }
115         }
```

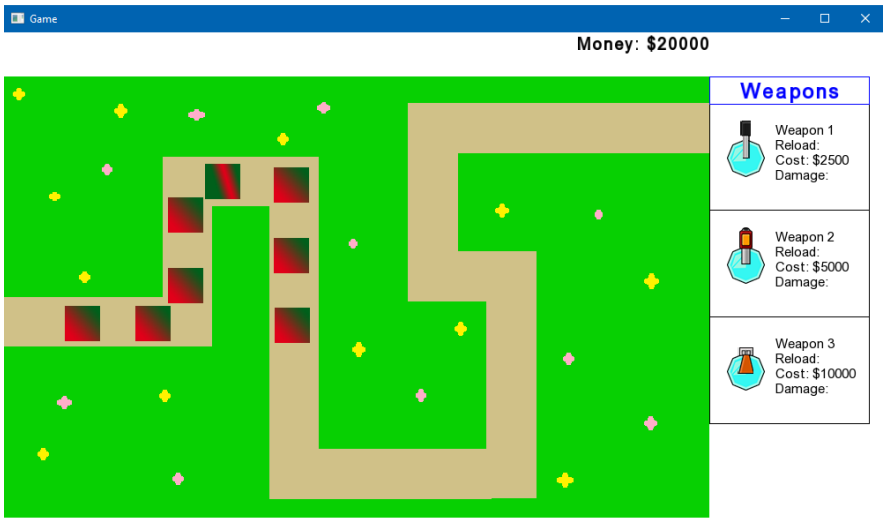
```
207         void update(sf::Time deltaTime)
208             {
209                 float time = deltaTime.asSeconds();
210                 for (int i = 0; i < totalEnemies; i++)
211                 {
212                     int move = EnemySpeed * time;
213                     //int value for move being the constant speed multiplied by the time per frame
214                     //was float, int made it uniform - finds floor keeps it constant
215                     if(Enemies[i].getPosition().x < 185 && Enemies[i].getPosition().y > 250 && Enemies[i].getPosition().y < 360)
216                         Enemies[i].setPosition(Enemies[i].getPosition().x + (move), Enemies[i].getPosition().y);
217                     else if(Enemies[i].getPosition().x > 175 && Enemies[i].getPosition().x < 205 && Enemies[i].getPosition().y > 150
218                             && Enemies[i].getPosition().y < 360)
219                         Enemies[i].setPosition(Enemies[i].getPosition().x, (Enemies[i].getPosition().y - (move)));
220                     else if(Enemies[i].getPosition().x > 175 && Enemies[i].getPosition().x < 305 && Enemies[i].getPosition().y > 130
221                             && Enemies[i].getPosition().y < 200)
222                         Enemies[i].setPosition(Enemies[i].getPosition().x + move, (Enemies[i].getPosition().y));
223                     else if(Enemies[i].getPosition().x > 305 && Enemies[i].getPosition().x < 315 && Enemies[i].getPosition().y > 130
224                             && Enemies[i].getPosition().y < 480)
225                         Enemies[i].setPosition(Enemies[i].getPosition().x, (Enemies[i].getPosition().y + (move)));
226                     else if(Enemies[i].getPosition().x > 300 && Enemies[i].getPosition().x < 550 && Enemies[i].getPosition().y > 470
227                             && Enemies[i].getPosition().y < 500)
228                         Enemies[i].setPosition(Enemies[i].getPosition().x + move, (Enemies[i].getPosition().y ));
229                     else if(Enemies[i].getPosition().x > 540 && Enemies[i].getPosition().x < 560 && Enemies[i].getPosition().y > 260
230                             && Enemies[i].getPosition().y < 500)
231                         Enemies[i].setPosition(Enemies[i].getPosition().x, (Enemies[i].getPosition().y - move ));
232                     else if(Enemies[i].getPosition().x > 470 && Enemies[i].getPosition().x < 560 && Enemies[i].getPosition().y > 240
233                             && Enemies[i].getPosition().y < 270)
234                         Enemies[i].setPosition(Enemies[i].getPosition().x - move, (Enemies[i].getPosition().y ));
235                     else if(Enemies[i].getPosition().x > 450 && Enemies[i].getPosition().x < 480 && Enemies[i].getPosition().y > 95
236                             && Enemies[i].getPosition().y < 270)
237                         Enemies[i].setPosition(Enemies[i].getPosition().x, (Enemies[i].getPosition().y - move));
238                     else
239                         Enemies[i].setPosition(Enemies[i].getPosition().x + move, (Enemies[i].getPosition().y));
240                     //Above code moves the enemy along based on what point on the map it is
241                 }
242             }
```

This concluded Iteration 2 making a functioning game whereby you could add weapons (with money being deducted) and enemies would be auto added upon the pressing the “Play Game” button with the enemies moving along the path.

Tests for Iteration 2:

	Problem Being Tested	Action to be taken to test this problem	Expected Result
5	Iteration 2 - Enemies and Enemy Animation	Click play button	Enemies added (not too many, not too little, must equal the current score of the player)
6		Click play button	Enemies added one after the other
7		Click play button	Enemies move one after the other following shape of map





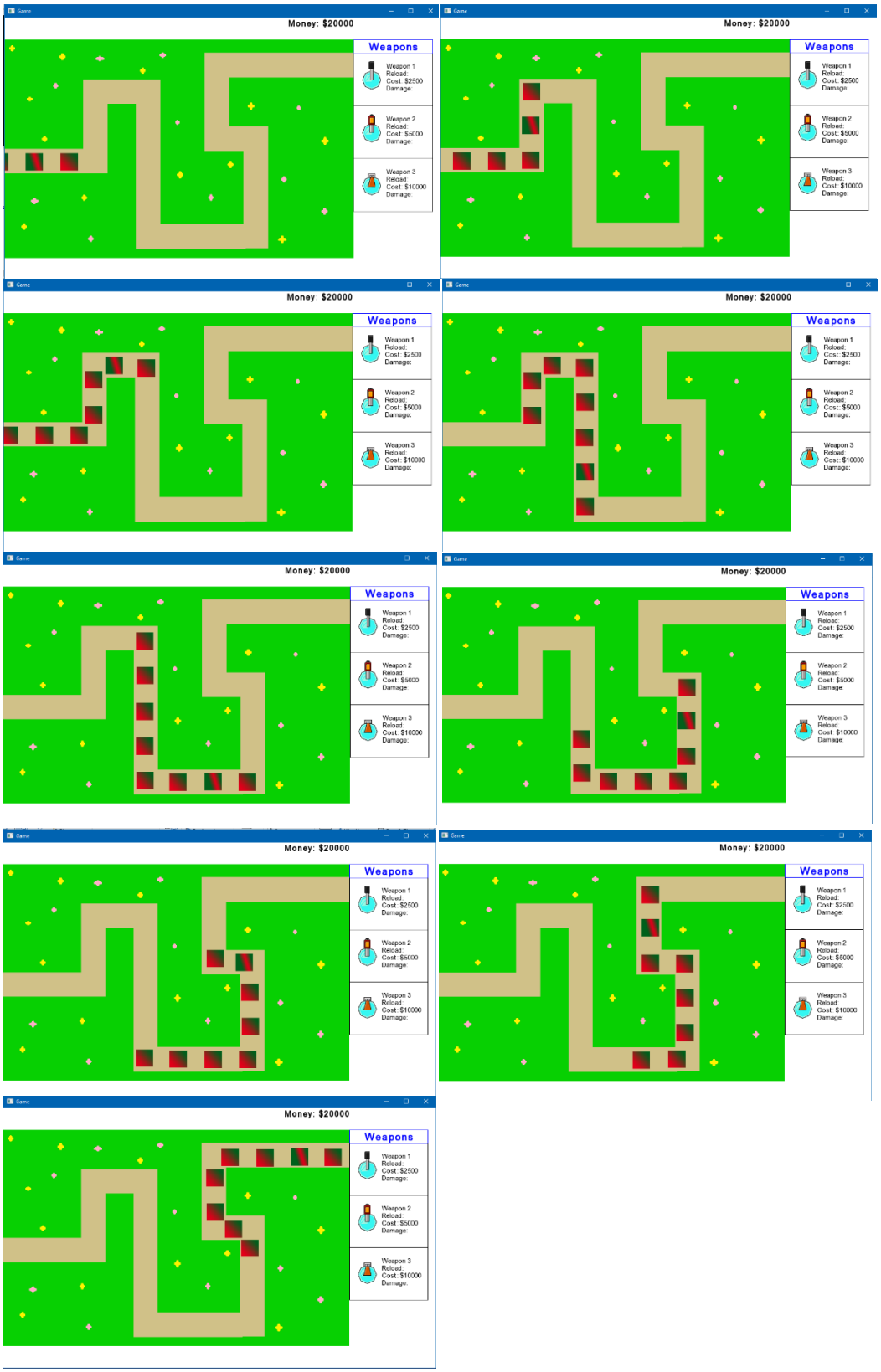
Enemy 1
10 Score

Enemy 2
50 Score

Enemy 3
100 Score

The initial score (not printed) is 170. As can be seen, the two attempts at playing the game give two different enemy combinations showing the random function is working. In the first one, we have two Enemy 1's, one Enemy 2 and one Enemy 3 (2* 10 + 1* 50 + 1*100 = 170) and in the second we have 7 Enemy 1's and one Enemy 3 = 170 as

Tests 2 and 3: The enemies should also follow the path and be roughly 100 pixels one after the other, and this can be seen to be true in the following screenshots:



And so all the tests have been passed for Iteration 2, we can move on to Iteration 3.

Iteration 3 – Weapon Rotation, Lock, Animation and Bullet Firing Modules

In Iteration 3, the purpose of this iteration is to force the weapon to rotate and follow a certain enemy, based upon which enemy it is locked upon. Once locked upon an enemy, if it is able to fire at that specific point in time (based upon its reload time) it will fire a shot, otherwise it will wait until it is able to fire a shot. In addition, the weapon should only lock on to an enemy and fire a shot at that enemy if the enemy is within the radius of the weapon (which is a 100pixel radius).

```
138             for (int i = 0; i < totalWeapons; i++)
139             {
140                 for (int j = 0; j < totalEnemies; j++)
141                 {
142                     //for every weapon and for every enemy, the following code needs to be run
143                     if (Weapons[i].isLocked())
144                     {
145                         float rotateBy = (atan((Weapons[i].getBasePosition().x - Enemies[Weapons[i].isLockedOn()].getPosition().x)
146                                               /(Weapons[i].getBasePosition().y - Enemies[Weapons[i].isLockedOn()].getPosition().y)))*(180/M_PI);
147                         if (rotateBy < 1)
148                         {
149                             rotateBy = rotateBy * (-1);
150                         }
151                         else
152                         {
153                             rotateBy = 180 - rotateBy;
154                         }
155                         Weapons[i].rotateTurret(rotateBy);
156                         //If the weapon is locked on to a certain enemy, then find the angle between the weapon and the enemy is is locked onto
157                         //using angles of a right angle triangle through arctan()
158                         //to make sure it follows at the correct angle, it is multiplied by -1 for values under 1
159
160                     }
161                     if (pow(pow((Weapons[i].getBasePosition().x + 21.5f) - (Enemies[j].getPosition().x), 2) +
162                             pow((Weapons[i].getBasePosition().y + 21.5f) - (Enemies[j].getPosition().y), 2), 0.5) <= 100)
163                         {
164                             Weapons[i].lock();
165                             Weapons[i].lockOnto(j);
166                         }
167                     else if (pow(pow((Weapons[i].getBasePosition().x + 21.5f) - (Enemies[j].getPosition().x + 40), 2) +
168                                  pow((Weapons[i].getBasePosition().y + 21.5f) - (Enemies[j].getPosition().y), 2), 0.5) <= 100)
169                         {
170                             Weapons[i].lock();
171                             Weapons[i].lockOnto(j);
172                         }
173                     else if (pow(pow((Weapons[i].getBasePosition().x + 21.5f) - (Enemies[j].getPosition().x), 2) +
174                                  pow((Weapons[i].getBasePosition().y + 21.5f) - (Enemies[j].getPosition().y), 2 + 40), 0.5) <= 100)
175                         {
176                             Weapons[i].lock();
177                             Weapons[i].lockOnto(j);
178                         }
179                     else if (pow(pow((Weapons[i].getBasePosition().x + 21.5f) - (Enemies[j].getPosition().x + 40), 2) +
180                                  pow((Weapons[i].getBasePosition().y + 21.5f) - (Enemies[j].getPosition().y), 2 + 40), 0.5) <= 100)
181                         {
182                             Weapons[i].lock();
183                             Weapons[i].lockOnto(j);
184                         }
185                     //Based upon the four corners approximation, if the enemy has entered the radius of the weapon, then lock on to it
186                     else
187                     {
188                         Weapons[i].unlock();
189                         std::cout << "Enemy " << j << " is not in the radius of Weapon " << i << std::endl;
190                         //If the enemy leaves the radius, then unlock the enemy from it
191                     }
192                 }
193             }
194         }
```

The above code was my initial idea for making the weapon turret follow the weapon. The idea was simple. Within the weapon class, I would have a Boolean value checking if the weapon was locked or not. In the case that it was, it would rotate the turret using simple trigonometry by working out the arctan of the opposite over the adjacent (with the opposite being the difference between the y values of the weapon and enemy and the adjacent being the difference between the x values of the weapon and enemy). If this angle was below 1, then I would multiply it by (-1) as the tan graph goes into negative angles too.

If it was not locked onto an enemy, it needed to find an enemy to lock onto. This would be done by checking if any of the enemies was within the radius of the circle, and if there was, then the algorithm would lock onto this enemy. Here I used an approximation as explained on my blog and approved by my stakeholders on the next page:

Ghusharib's Computing Coursework – Tower Defence Game

Blog related to my Computing Coursework

HOME

Weapon Radius Approximation

NOVEMBER 28, 2017 ~ GHUSHARIBCOMPUTINGGAME

When an enemy enters a weapon's radius, the weapon needs to know that the enemy is there so that it can lock onto it. However, this is a fairly tough and time consuming algorithm to implement, as you are basically trying to find the minimum distance from a point to a square. However, there is a more efficient alternative I could implement, which I called the "Four Corners" Algorithm.

Basically, my circle (radius) in relation to my square (the enemy) is far bigger. What this means is that I can assume that one of the four corners of my square has entered the weapon radius the first time it enters the radius. In reality, this is not true but the enemy does not need to move too big a distance before this condition is met, and therefore, it is reasonable to approximate the four corners as entering the circle and that means the program only needs to make four checks. This approximation can be seen below:

Image 1 shows when it should register the enemy while Image 2 shows when it will register the enemy. Note it hasn't moved much.

Please can you reply to tell me whether or not you are happy with this approximation. Thank You!

8 thoughts on "Weapon Radius Approximation"

Nathan
NOVEMBER 28, 2017 AT 2:53 PM

Ghusharib, I agree with the content on this page.

★ Like

Reply

walesowilsonsepq
NOVEMBER 28, 2017 AT 3:02 PM

I am happy with this approximation, but can you increase the shooting radius very slightly to compensate for this? I think that it is important to ensure that the game is well balanced. Oli

★ Liked by 1 person

Reply

ghusharibcomputinggame
NOVEMBER 28, 2017 AT 9:21 PM

I can understand why you would want a slightly bigger radius but as you can see in my reply to Vinayak's Comment, there really is no cause for concern as the circle is huge in relation to the square and so will not cause a noticeable difference to any gameplay

★ Like

Reply

Praveen Murugathas
NOVEMBER 28, 2017 AT 9:26 PM

It's Fine

★ Like

Reply

Miron
NOVEMBER 28, 2017 AT 7:08 PM

I am happy with this approximation

★ Like

Reply

Vinayak Shastri
NOVEMBER 28, 2017 AT 9:11 PM

I am finding this approximation to be extremely pleasant, this approximation is incredibly ingenious in thought and it has been an honour to be a witness for this implementation. However, my main concern is that for a tower defence game, the importance of quick detection of enemy units is paramount, so I would like to receive a clarification as to what specifications these squares are in terms of size, as size matters, and to what degree of accuracy is the approximation numerically accounted for?

★ Liked by 1 person

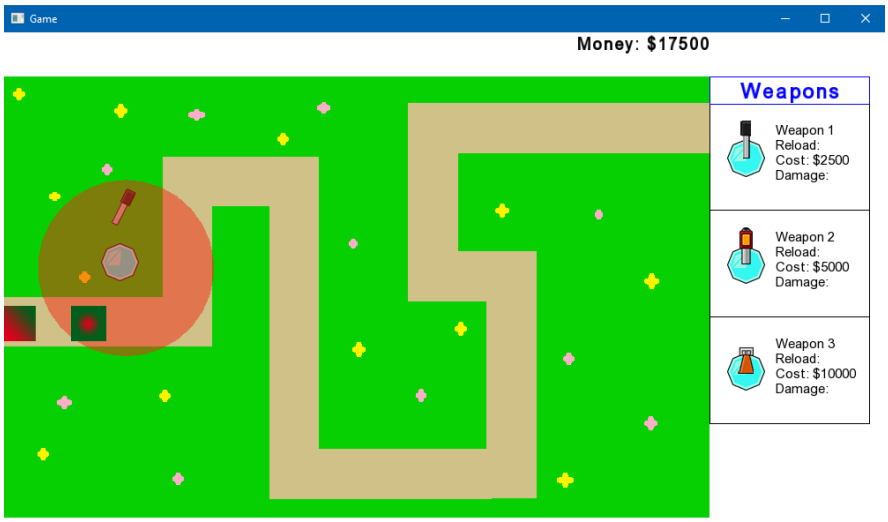
Reply

ghusharibcomputinggame
NOVEMBER 28, 2017 AT 9:19 PM

Thanks for your valuable comment. In regards to your concerns, Weapons 1 and 2 have a radius of 100pixels each and Weapon 3 has a radius of 175 pixels. That makes Weapons 1 and 2 200 pixels wide while Weapon 3 is 350 pixels wide. The enemy, however, is only 40pixels by 40pixels. What this means is that the enemy only needs to go inside the circle (in the worst case scenario) a maximum of a few (about 2) pixels to be recognised as having a corner within the radius and therefore is not a major cause for concern. In terms of accuracy, I have no percentage but I can assure you it will work well enough for you to not notice the difference.

This approximation therefore helped me decide whether or not an enemy was within the radius of my circle. This approximation worked perfectly well, but other things not so well.

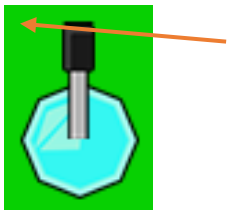
Firstly, the anchor point of rotation for my weapon turret was completely wrong. This meant that the turret rotated from the wrong point of rotation and therefore had a weird look to it where it appeared detached from the base:



N.b. the red circle was for debugging purposes.

This was something that needed to be fixed. In addition, the angle of rotation was completely wrong and the weapon was not following the enemy at all, but appeared to be completely random.

I had to fix this, but how? Firstly, I needed to adjust the anchor point. The anchor point is something that is predefined by SFML as being in a specific point for each and every sprite, namely here:

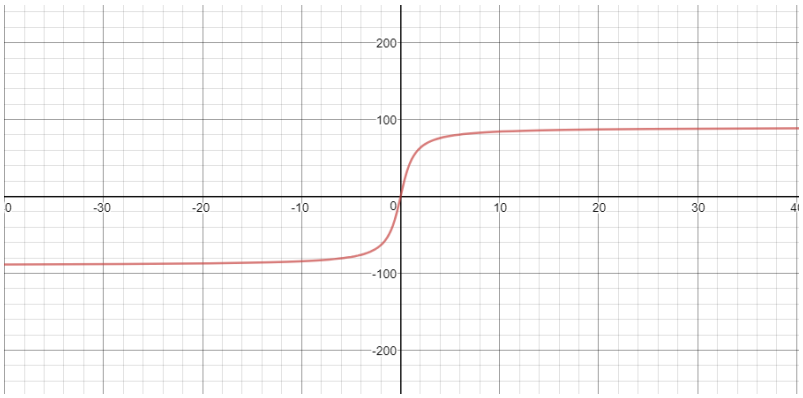


This can be easily fixed. All I need to do is simply move along the anchor point half the texture size along and the full texture size down to make it universal for all the three kinds of turrets regardless of their size:

```
25      void setTurretPosition(float x, float y)
26      {
27          TurretSprite.setOrigin((TurretTexture.getSize().x/2), TurretTexture.getSize().y);
28          TurretSprite.setPosition(x + (TurretTexture.getSize().x/2), y + TurretTexture.getSize().y);
29      }
```

When setting the turret position, the turret's anchor point is also set to make it an automatic feature.

The next obstacle to tackle is the rotation angle. This comes down to understanding the 4 cases that can occur while the game is running:



The graph of arctan(x)

Case 1)

The enemy is above and to the right of the weapon. In this case, the x distance is positive but the y distance is negative and so you will get a negative value of opposite/adjacent. This will give you the negative value of the angle when you do arctan(op/adj). In addition, you need to do 90 minus this to get the right angle and so the formula for this will be 90 – ((-1)*arctan(op/adj))

Case 2)

The enemy is below and to the right of the weapon. In this case, the x distance and the y distance is positive. However, the angle is 90 degrees off and so you will need to do 90 + arctan(op/adj).

Case 3)

The enemy is below and to the left of the weapon. In this case, the x distance is negative and the y distance is positive. This gives a negative value and in addition, the angle is 180 degrees off and 90 minus the real angle. Therefore, here we need to do 180 + (90 – ((-1)*arctan(opp/adj)))

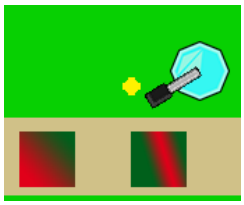
Case 4)

The enemy is above and to the left of the weapon. In this case, both the x and y distances are negative and so you actually get a positive value. However, this angle is 270 degrees off so you need to do 270 + arctan(opp/adj).

And so we get better code which is as follows:

```
138             for (int i = 0; i < totalWeapons; i++)
139             {
140                 for (int j = 0; j < totalEnemies; j++)
141                 {
142                     if (Weapons[i].isLocked())
143                     {
144                         float rotateBy = (atan((Enemies[Weapons[i].isLockedOn()].getPosition().y - Weapons[i].getBasePosition().y)/
145                                               (Enemies[Weapons[i].isLockedOn()].getPosition().x - Weapons[i].getBasePosition().x)))*(180/M_PI);
146                         if (Weapons[i].getBasePosition().y > Enemies[Weapons[i].isLockedOn()].getPosition().y
147                             && Weapons[i].getBasePosition().x < Enemies[Weapons[i].isLockedOn()].getPosition().x)
148                         {
149                             rotateBy = 90 - ((-1) * rotateBy);
150                         }
151                         else if (Weapons[i].getBasePosition().y < Enemies[Weapons[i].isLockedOn()].getPosition().y
152                             && Weapons[i].getBasePosition().x < Enemies[Weapons[i].isLockedOn()].getPosition().x)
153                         {
154                             rotateBy = 90 + rotateBy;
155                         }
156                         else if(Weapons[i].getBasePosition().y < Enemies[Weapons[i].isLocked()].getPosition().y
157                                 && Weapons[i].getBasePosition().x > Enemies[Weapons[i].isLockedOn()].getPosition().x)
158                         {
159                             rotateBy = 180 + (90 - (rotateBy * (-1)));
160                         }
161                         else if (Weapons[i].getBasePosition().y > Enemies[Weapons[i].isLocked()].getPosition().y
162                                 && Weapons[i].getBasePosition().x > Enemies[Weapons[i].isLockedOn()].getPosition().x)
163                         {
164                             rotateBy = (270 + rotateBy);
165                         }
166                         Weapons[i].rotateTurret(rotateBy);
167                     }
168                     if (pow(pow((Weapons[i].getBasePosition().x + 21.5f) - (Enemies[j].getPosition().x), 2) +
169                             pow((Weapons[i].getBasePosition().y + 21.5f) - (Enemies[j].getPosition().y), 2), 0.5) <= 100)
170                     {
171                         Weapons[i].lock();
172                         Weapons[i].lockOnto(j);
173                     }
174                     else if (pow(pow((Weapons[i].getBasePosition().x + 21.5f) - (Enemies[j].getPosition().x + 40), 2) +
175                             pow((Weapons[i].getBasePosition().y + 21.5f) - (Enemies[j].getPosition().y), 2), 0.5) <= 100)
176                     {
177                         Weapons[i].lock();
178                         Weapons[i].lockOnto(j);
179                     }
180                     else if (pow(pow((Weapons[i].getBasePosition().x + 21.5f) - (Enemies[j].getPosition().x), 2) +
181                             pow((Weapons[i].getBasePosition().y + 21.5f) - (Enemies[j].getPosition().y), 2 + 40), 0.5) <= 100)
182                     {
183                         Weapons[i].lock();
184                         Weapons[i].lockOnto(j);
185                     }
186                     else if (pow(pow((Weapons[i].getBasePosition().x + 21.5f) - (Enemies[j].getPosition().x + 40), 2) +
187                             pow((Weapons[i].getBasePosition().y + 21.5f) - (Enemies[j].getPosition().y), 2 + 40), 0.5) <= 100)
188                     {
189                         Weapons[i].lock();
190                         Weapons[i].lockOnto(j);
191                     }
192                     else
193                     {
194                         Weapons[i].unlock();
195                     }
196                 }
197             }
198         }
```

This fixes the past problems:



However, looking at the above weapon a new problem has arose. The weapons are constantly unlocking and therefore end up relocking on the last enemy rather than the first enemy that arrives into its radius. This causes problems as the weapon should only lock onto one enemy and not lock onto another until the enemy leaves its radius.

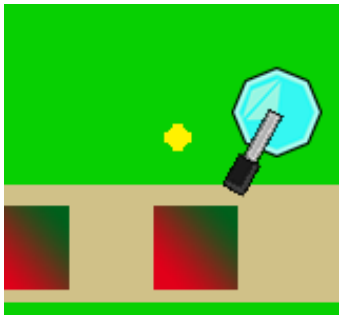
The reason this happens comes down to the loops. The loop for checking the enemies runs within the weapons loop every time. This means that for every enemy, if the weapon is not locked onto that specific enemy, the weapon unlocks. Instead, the weapon should not compare itself to other enemies until the enemy it is locked on has left its radius, and so the for loop should instead be moved to a different point entirely, as is done on the next page:

Firstly however, I need to move checking if an enemy is in range into a separate function itself. This is because I will now need to call it from more than one position in my code, and so it is better for it to be a self-contained unit rather than multiple copied code.

```
187  bool isInRange(sf::Vector2f WeaponPosition, sf::Vector2f EnemyPosition)
188  {
189      if (pow(pow((WeaponPosition.x + 21.5f) - (EnemyPosition.x), 2) + pow((WeaponPosition.y + 21.5f) - (EnemyPosition.y), 2), 0.5) <= 100)
190          {
191              return true;
192          }
193      else if (pow(pow((WeaponPosition.x + 21.5f) - (EnemyPosition.x + 40), 2) + pow((WeaponPosition.y + 21.5f) - (EnemyPosition.y), 2), 0.5) <= 100)
194          {
195              return true;
196          }
197      else if (pow(pow((WeaponPosition.x + 21.5f) - (EnemyPosition.x), 2) + pow((WeaponPosition.y + 21.5f) - (EnemyPosition.y), 2 + 40), 0.5) <= 100)
198          {
199              return true;
200          }
201      else if (pow(pow((WeaponPosition.x + 21.5f) - (EnemyPosition.x + 40), 2) + pow((WeaponPosition.y + 21.5f) - (EnemyPosition.y), 2 + 40), 0.5) <= 100)
202          {
203              return true;
204          }
205      else
206          {
207              return false;
208          }
209  }
```

The above code does exactly the same function as before but is now separated into its own function.

```
138              for (int i = 0; i < totalWeapons; i++)
139              {
140                  if (Weapons[i].isLocked())
141                      {
142                          float rotateBy = (atan((Enemies[Weapons[i].isLockedOn()].getPosition().y - Weapons[i].getBasePosition().y)/
143                                                 (Enemies[Weapons[i].isLockedOn()].getPosition().x - Weapons[i].getBasePosition().x)))*(180/M_PI);
144                          if (Weapons[i].getBasePosition().y > Enemies[Weapons[i].isLockedOn()].getPosition().y
145                              && Weapons[i].getBasePosition().x < Enemies[Weapons[i].isLockedOn()].getPosition().x)
146                          {
147                              rotateBy = 90 - ((-1) * rotateBy);
148                          }
149                          else if (Weapons[i].getBasePosition().y < Enemies[Weapons[i].isLockedOn()].getPosition().y
150                              && Weapons[i].getBasePosition().x < Enemies[Weapons[i].isLockedOn()].getPosition().x)
151                          {
152                              rotateBy = 90 + rotateBy;
153                          }
154                          else if(Weapons[i].getBasePosition().y < Enemies[Weapons[i].isLocked()].getPosition().y
155                                  && Weapons[i].getBasePosition().x > Enemies[Weapons[i].isLockedOn()].getPosition().x)
156                          {
157                              rotateBy = 180 + (90 - (rotateBy * (-1)));
158                          }
159                          else if (Weapons[i].getBasePosition().y > Enemies[Weapons[i].isLocked()].getPosition().y
160                                   && Weapons[i].getBasePosition().x > Enemies[Weapons[i].isLockedOn()].getPosition().x)
161                          {
162                              rotateBy = (270 + rotateBy);
163                          }
164                          Weapons[i].rotateTurret(rotateBy);
165
166                          if(!isInRange(Weapons[i].getBasePosition(), Enemies[Weapons[i].isLockedOn()].getPosition()))
167                          {
168                              Weapons[i].unlock();
169                          }
170                      }
171                      else
172                      {
173                          for (int j = 0; j < totalEnemies; j++)
174                          {
175                              if(isInRange(Weapons[i].getBasePosition(), Enemies[j].getPosition()))
176                              {
177                                  Weapons[i].lockOnto(j);
178                              }
179
180                          }
181                      }
182                  }
183              }
```



This now works well as shown. It is now aimed at the correct enemy this time

The final stage was making the bullet of the weapon fire at the correct angle at the enemy that the weapon was locked onto (and not any other enemy). This was easy and did not cause any problems.

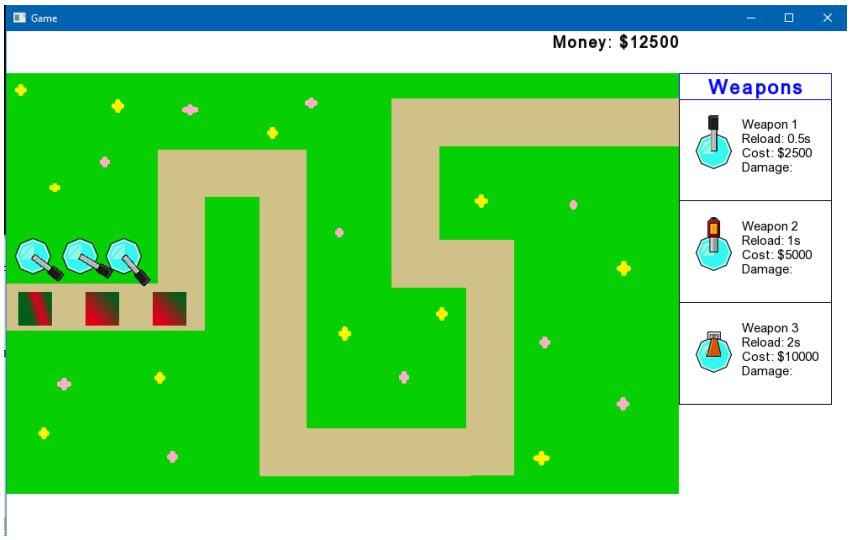
```
193                     if(Weapons[i].fireStatus())
194                         {
195                             fireBullet(i);
196                             if (hitTarget(i, Weapons[i].isLockedOn()))
197                                 Weapons[i].reset();
198                             //If the weapon is firing (i.e. its status is ready) then fire the bullet.
199                             //If it is hit, then unlock the weapon
200                         }
201                     else if (Weapons[i].isReadyToFire() && Weapons[i].isLocked())
202                         {
203                             std::cout << "FIRE!" << std::endl;
204                             Weapons[i].reset();
205                             Weapons[i].restartTimer();
206                             Weapons[i].fired();
207                             fireBullet(i);
208                             //When the weapon is ready to fire, fire the bullet at the enemy it is locked on
209                         }
```

```
213 bool hitTarget(int i, int j)
214 {
215     if (Weapons[i].getBulletPosition().x > Enemies[j].getPosition().x  - 20 && Weapons[i].getBulletPosition().x < Enemies[j].getPosition().x + 60
216         && Weapons[i].getBulletPosition().y > Enemies[j].getPosition().y - 20 && Weapons[i].getBulletPosition().y < Enemies[j].getPosition().y + 60)
217         {
218             lockToggle = false;
219             //If the bullet has hit the enemy, then return true and proceed to fire again
220             return true;
221         }
222     else
223         return false;
224 }
225
226 void fireBullet(int weapon)
227 {
228     Weapons[weapon].moveBullet(Weapons[weapon].getBulletDirection().x, Weapons[weapon].getBulletDirection().y);
229     //Move the weapon towards the enemy's position
230 }
```

Tests for Iteration 3:

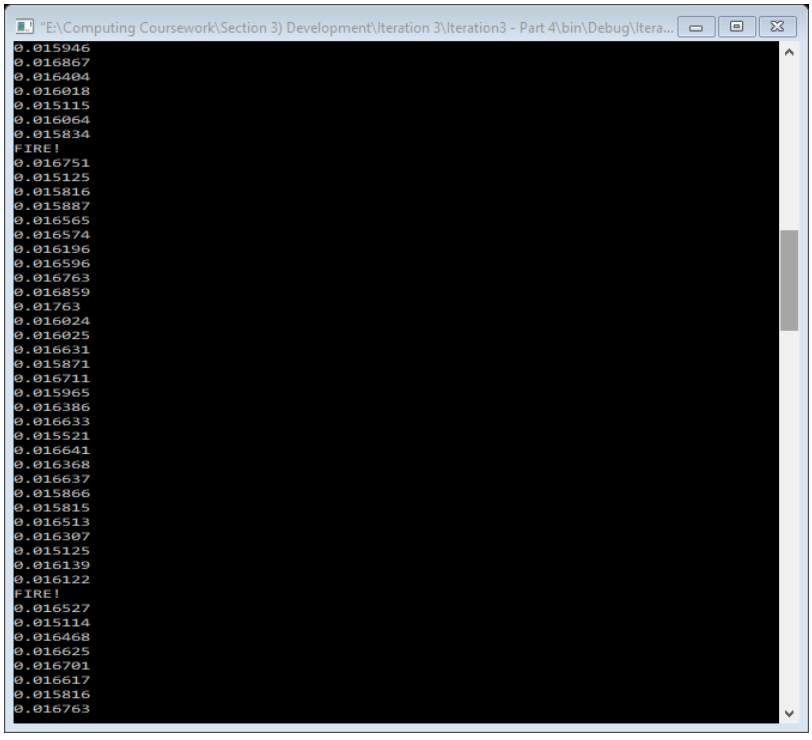
	Problem Being Tested	Action to be taken to test this problem	Expected Result
8	Iteration 3 - Weapons Animations	Click play button	Weapons follow enemy once the enemy is within their radii.
9		Click play button	Weapons fire at enemy when enemy is within their radii and have a reset time before firing again (make sure these times are followed)

Test 1)



The three weapons above are pointing at the enemy that they are locked onto only, and not any other weapon. This remains like this throughout, and so the test is passed.

Test 2)

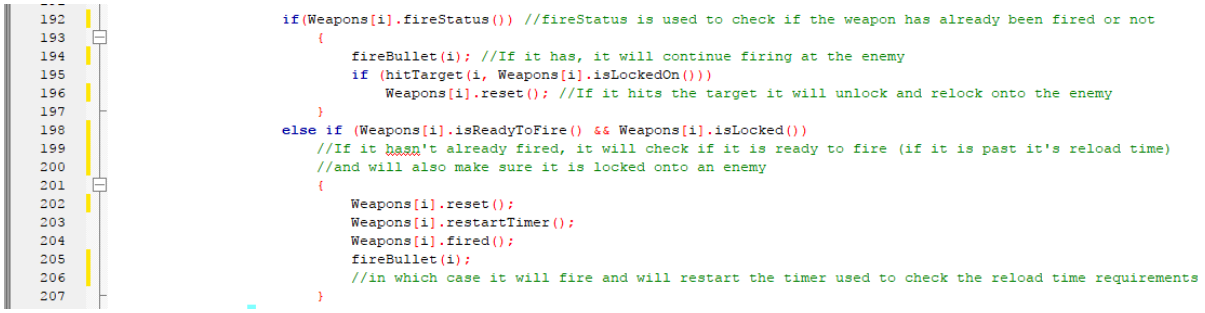


The time passed between successive "Fire!" calls adds to 30 seconds. Therefore, this works

Iteration 3 has finished whereby a working program of Weapons added follow enemies added and fire shots at them and has passed all the tests. I can therefore progress to Iteration 4, noting that my Hit Target function will come in handy for it.

Iteration 4 – Collision Module, Enemy Death Module, Bullet Hit Module

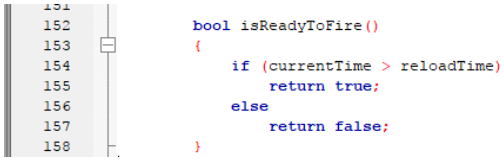
So at this point, I now have a game with the ability to place weapons and enemies are randomly generated and placed into the game based on the player's score. However, bullets now need to be fired from the weapon at the enemy and the bullet needs to hit the enemy. Upon colliding with the enemy, the enemy's health should decrease and once the enemy's health is equal to or less than zero, the enemy should be removed from the screen.



```
192             if(Weapons[i].fireStatus()) //fireStatus is used to check if the weapon has already been fired or not
193                 {
194                     fireBullet(i); //If it has, it will continue firing at the enemy
195                     if (hitTarget(i, Weapons[i].isLockedOn()))
196                         Weapons[i].reset(); //If it hits the target it will unlock and relock onto the enemy
197                 }
198             else if (Weapons[i].isReadyToFire() && Weapons[i].isLocked())
199                 //If it hasn't already fired, it will check if it is ready to fire (if it is past it's reload time)
200                 //and will also make sure it is locked onto an enemy
201                 {
202                     Weapons[i].reset();
203                     Weapons[i].restartTimer();
204                     Weapons[i].fired();
205                     fireBullet(i);
206                     //in which case it will fire and will restart the timer used to check the reload time requirements
207                 }
```

The above code is what I used to fire bullets at enemies. If the weapon has not already fired a bullet (and the bullet has not yet reached the enemy) then it will be fired and the timer in regards to the reload time will be restarted. This is to ensure that the weapon does not fire again unless it has reached its reload time. It will also set the isFired variable to true so that the weapon will continue firing at the weapon it is locked onto until the bullet has hit its intended target. This is to ensure that the weapon doesn't end up locking onto another enemy while it is still firing a bullet (and hence the bullet ends up never reaching the enemy is was intended to hit).

For the isReadyToFire() method, the following code was implemented



```
152 bool isReadyToFire()
153 {
154     if (currentTime > reloadTime)
155         return true;
156     else
157         return false;
158 }
```

It is a very simple piece of code whereby if the time the weapon has been waiting for is more than the reload time, it will say it is ready to fire, else it will say it is not.

The following code caused a lot of problems as shown in my problem log for multiple reasons. It was causing my game to occasionally crash when attempting to remove an enemy, and required a lot of debugging.

```
210             for (int i = 0; i < totalEnemies; i++)
211             {
212                 if (Enemies[i].isDead())
213                 {
214                     //delete &Enemies[i];
215                     Enemies.erase(Enemies.begin() + i);
216                     //Removes the enemy from the vector
217                     totalEnemies--;
218                     //Decreases the totalEnemies to ensure the game doesn't bug out
219                     for (int x = 0; x < totalWeapons; x++)
220                     {
221                         Weapons[x].unlock();
222                         Weapons[x].reset();
223                         //It then proceeds to unlock all the weapons so that no weapon remains locked onto an enemy which has died
224                         //and so the game does not crash
225                     }
226                     i = 0;
227                     //The counter for the for loop is reset to ensure an enemy is not missed out
228                 }
229             }
230
231             if (totalEnemies == 0)
232             {
233                 playingGame = false;
234             }
```

As can be seen, delete &Enemies[i] is commented out. This was part of the debugging process whereby I was trying to work out which line of code was causing my game to crash.

The delete function deletes an object from the computer's memory using the pointer to the object (hence the & before Enemies[i]). This I thought was very important to avoid overloading the user's RAM, but apparently was unnecessary and causing me problems. This was because when the object was proceeded to be removed from the enemy's vector on line 215, the object would not be found by the program. This was because the object had been deleted, but this was required to be removed from the vector otherwise that too would cause the game to crash. I was therefore left in a state whereby I could not put either line of code before the other and had to live with only one.

However, when I added enemies to the vector, I used the Enemies.push_back() function. This means that when I use Enemies.erase(), not only is the object removed from the vector, it is also deleted from the user's RAM. This meant that the delete function was unnecessary and so I could live with only using the erase function.

There were a few things which could have caused the game to crash, and to overcome these, I unlocked all weapons when an enemy died (to ensure that a weapon did not continue to fire at an enemy did not exist) and also made sure every enemy was checked. This did not seem to cause any further problems so at this stage I was happy.

A few things to note are my hitTarget() and fireBullet() functions:

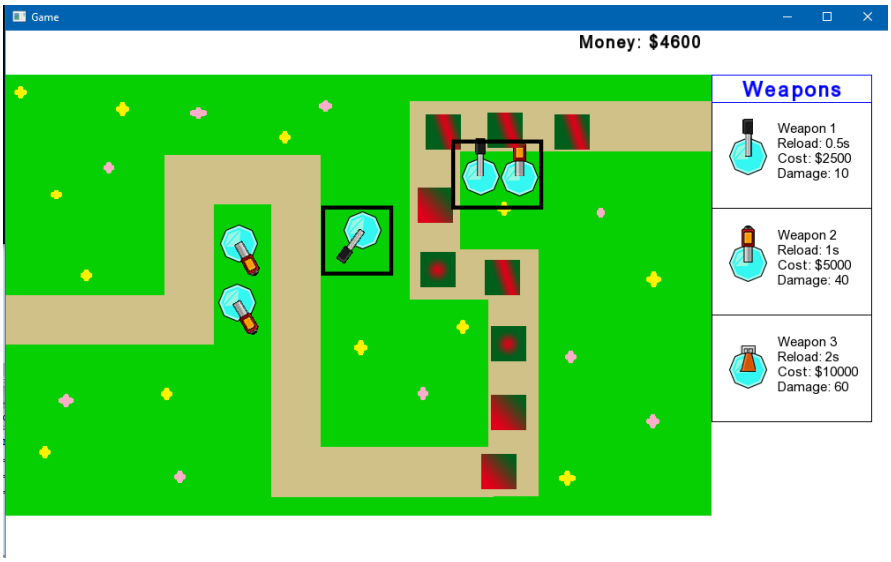
```
237     bool hitTarget(int i, int j)
238     {
239         if (Weapons[i].getBulletPosition().x > Enemies[j].getPosition().x  - 20 && Weapons[i].getBulletPosition().x < Enemies[j].getPosition().x + 60
240             && Weapons[i].getBulletPosition().y > Enemies[j].getPosition().y - 20 && Weapons[i].getBulletPosition().y < Enemies[j].getPosition().y + 60)
241             { //If the bullet is on the enemy's texture
242                 lockToggle = false;//Then trigger the lock toggle
243                 Enemies[j].reduceHealth(Weapons[i].getDamage()); //Reduce the enemy's health
244                 return true; //And inform the requester that the enemy has now been hit (and it is free to do whatever)
245             }
246         else
247             return false;
248     }
249
250     void fireBullet(int weapon)
251     {
252         Weapons[weapon].moveBullet(Weapons[weapon].getBulletDirection().x, Weapons[weapon].getBulletDirection().y);
253         //Move the bullet in the direction of the enemy it is locked onto
254     }
```

For the next step, I simply had the score and money of the user increase whenever an enemy dies using the simple code:

```
213                 for (int i = 0; i < totalEnemies; i++)
214                 {
215                     if (Enemies[i].isDead())
216                     {
217                         Score = Score + Enemies[i].getScore();
218                         Money = Money + Enemies[i].getReturnMoney() * 2;
219                         Enemies.erase(Enemies.begin() + i);
220                         totalEnemies--;
221                         std::cout << "Enemy " << i << " was Removed. Enemies left: " << totalEnemies << std::endl;
222                         i = 0;
223                     }
224                 }
```

So the score is increased according to the enemy's score and the money is increased by the return money of the enemy multiplied by 2. Note that I no longer reset the weapons because this was not in reality causing the game to crash. The game also reduces the health of the tower once the enemies reach the end of the path.

However, a problem that did arise was the weapons no longer recognising enemies as being in their radii. This was very confusing as back in Iteration 3, it seemed it was working perfectly fine. The problem is shown below:



These weapons should be pointing at the corresponding enemies, but they clearly are not

The problem came down to the lock toggle. The lock toggle's intention was to keep the weapon locked onto an enemy until the bullet it had fired had not yet hit the enemy. However, this was causing major problems for it, and was not being reset to false when it should have been. Therefore, in the original code, when it checked to see if the enemy was in range and if lock toggle was off:

```
189         for (int j = 0; j < totalEnemies; j++)
190         {
191             if(isInRange(Weapons[i].getBasePosition(), Enemies[j].getPosition())) && !lockToggle)
192             {
193                 Weapons[i].lockOnto(j);
194                 lockToggle = true;
195             }
```

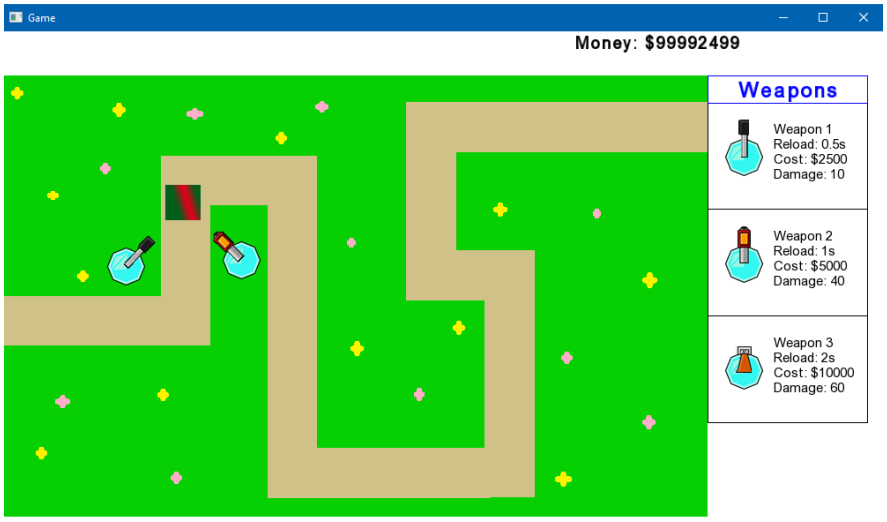
The program would recognise the conditions as not being met. This meant that the weapon would no longer lock onto any other enemy any more. It would be frozen and hence show the bug noted before. The correct method to follow here would be to remove the lock toggle completely, which is exactly what I did. I would have thought this would have caused problems, but in reality the code already ensured that the weapon was locked onto an enemy until it went outside of its radius and/or died. Therefore, the lockToggle had been useless and had been causing unnecessary problems, and it was good that I had gotten rid of it.

10	Iteration 4 - Collisions	Click play button	Weapons fire at enemy once the enemy is within their radii causing enemy health to decrease. If health of enemy is 0, enemy is removed from screen, score and money increased
11		Click play button	Enemy reaching end of path attack the tower causing tower health to decrease
12		Let enemies attack	If tower health is 0, game ends and message appears

Tests for Iteration 4:

Test 1) Weapons attack Enemies

The weapon can be seen to be shooting at the enemy in this screen



The second weapon also joins in with the attack at this point

Very quickly, the weapons manage to reduce the health of the enemy. This causes the enemy to be removed from the screen and for there to be an increase in the money the user has. The score has also increased although it is not visible.

Test 2) No weapons are placed and the enemies approach the end of the map. Upon reaching the end, the console log outputs the tower health which can be seen to go down by

125 both times (from 500) as the two attacking enemies are the same. This will be shown in a health bar at a later stage and the game over screen.

Iteration 5 – Powerup Class, Used Module, Destroy All Enemies Module

In this iteration, the powerup is implemented so that it can only be bought once and used once. Upon use, it will clear all enemies from the screen.

I first implemented a use powerup button to be pressed whenever the user requires to use the powerup. I made this cost $15000 to correspond with the user requirements of it being expensive enough to deter people from using it.

```
167                 case sf::Event::MouseButtonPressed:
168                     if (event.mouseButton.button == sf::Mouse::Left)
169                     {
170                         if(!PowerUp.isUsed() && Money - PowerUp.getCost() >= 0 &&
171                            event.mouseButton.x > PowerUp.getButtonPosition().x && event.mouseButton.x < PowerUp.getButtonPosition().x + 90
172                            && event.mouseButton.y > PowerUp.getButtonPosition().y && event.mouseButton.y < PowerUp.getButtonPosition().y + 50)
173                            //If user has used left mouse button to click on the powerup button and the powerup has not already been used
174                            //and the user can afford the powerup
175                         {
176                             usingPowerUp = true;
177                             //Then the powerup is being used
178                             PowerUp.use();
179                             //set the is used value to true
180                             Money = Money - PowerUp.getCost();
181                             //And remove the money from the user's money balance
182                         }
183                     }
```

The above code checks to see if the user has used the powerup already or not. If they have, then the powerup cannot be used and so nothing happens. If they haven't, and they can afford the powerup, the money is deducted from their balance, the variable showing whether or not the powerup has been used is updated to the value "true" and the power up animation will now play. This is done using the code below:

```
511         void runPowerupAnimation(float time)
512         {
513             PowerUp.setPosition(400, 300);
514             powerupAnimationTimer += time;
515             //The powerup is placed in the middle of the screen (an image of a bomb) and the timer for it is
516             //increased according to the time since the last update
517             if(powerupAnimationTimer >= 0.25)
518             {
519                 powerupAnimationTimer = 0;
520                 currentPowerUpTexture++;
521                 powerupAnimationSprite.setTextureRect(sf::IntRect(currentPowerUpTexture * 800, 0, 800, 500));
522                 //If the timer has gone past a predefined limit, the next texture of the animation is displayed
523             }
524             if(currentPowerUpTexture == 5)
525             {
526                 for(int i = 0; i < totalEnemies; i++)
527                 {
528                     Score = Score + Enemies[i].getScore();
529                     Money = Money + Enemies[i].getReturnMoney() * 5;
530                     Enemies.erase(Enemies.begin() + i);
531                     //if we have reached the last texture, the enemies are all removed
532                     //The money from killing the enemies is also given to the user
533                 }
534                 totalEnemies = 0;
535                 //The total enemies now becomes 0
536                 PowerUp.resetPosition();
537                 usingPowerUp = false;
538                 //The powerup's position is reset to it's original position (in the weapons menu) and the
539                 //powerup is no longer in use so usingPowerUp is set to false
540             }
541         }
```

The code updates the powerup animation to display the next texture every 0.25 seconds and once it is on its last texture, it removes all the enemies from the screen and returns all the money you would have got had you killed the enemies using normal weapons. The powerup is then placed back into the weapons bar on the right.

The animation is a key part of the game and requires some interesting and complex

code. Firstly, I had to create a “sprite sheet” for the animation, so that the animation would run through different sections of the sprite sheet:

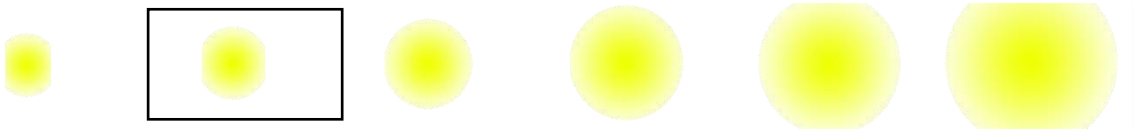
I now had to make my code shift through these animations one by one. As can be noted in the above code, I had two things to help me with this. One was the currentPowerUpTexture integer variable. The other was the powerupAnimationTimer. The powerupAnimationTimer would take in a float value of the time since the last update (fed in by the SFML library). Every 0.25 seconds that passed, the currentPowerUpTexture would increment by 1 and the powerUpAnimationTimer would reset to 0.

To make the animation run, I would use the SFML getTextureRect() function. This uses another SFML feature known as sf::intRect to only show a defined rectangle of pixels of the texture. To use this, you require in your intRect():

1. The starting x position
2. The starting y position
3. The width of each texture in pixels
4. The length of each texture in pixels

Requirements 2, 3 and 4 were all constants. The starting y position would always be zero (As we would start from the top of the texture and draw the texture on downwards) and the width and length of each sprite on the sprite sheet was 800x500 pixels. However, the starting x position was dependent on the currentPowerUpTexture. If this was 0, it would start from 0 * 800 = 0 pixels. If it was 1, it would start from 1 * 800 = 800 pixels and so on. This meant that the first animation would show this:

And the next would show this:



And so on.

The last part was removing all the enemies. I simply used the myVector.erase() function which was used back in Iteration4 to empty out the vector one by one and get the return money from each enemy as well. The game would then loop back.

Tests for Iteration 5:

	Problem Being Tested	Action to be taken to test this problem	Expected Result
13	Iteration 5 - Powerup (and corresponding Collisions	Click and drag powerup during simulation if not already used once	Powerup added to part of screen and money is deducted from total money
14		Powerup attacks enemy	Enemy health decreases
15		Click and drag powerup during simulation if already used once	Error message is displayed
16		Click and drag normal weapon during simulation	Error message is displayed

The click and drag feature is not relevant for this iteration anymore. This is because at the time of writing, it had not been decided what the powerup should be but now we have decided it is a bomb that kills all enemies on screen, it has become obvious that I do not require a click and drag feature for it. In fact, the above tests are all irrelevant and instead I need to only test two things:

1) Testing the animation: For this test, I had to reduce the animation timer to update every 2 seconds instead to get all the frames.
2) All the enemies on the screen should disappear

The above screenshots show the animation taking place and that after the animation has occurred, no enemies are visible and the money has increased. Iteration 5 is complete with no problems and has passed all tests.

Iteration 6 – Bug removal and halfway user feedback

Iteration 6 was meant to loop the game simulation and weapon placement. However, if you go back to iteration 4, you will notice this was already done when I check if the totalEnemies is 0, and if it is, the game is no longer playing and so users are able to place weapons once again. Therefore, it seemed unnecessary to follow iteration 6 as it was written. Instead, I decided to take this opportunity to address some features I may have forgotten in the planning stage, remove bugs and get some feedback on the progress on my game so far (and make any suggested improvements).

1) Enemy Movement in the update() function

Back in Iteration 2, I created an update function to move my enemies along the path. It looked a bit like this:

```
207         void update(sf::Time deltaTime)
208             {
209                 float time = deltaTime.asSeconds();
210                 for (int i = 0; i < totalEnemies; i++)
211                 {
212                     int move = EnemySpeed * time;
213                     //int value for move being the constant speed multiplied by the time per frame
214                     //was float, int made it uniform - finds floor keeps it constant
215                     if(Enemies[i].getPosition().x < 185 && Enemies[i].getPosition().y > 250 && Enemies[i].getPosition().y < 360)
216                         Enemies[i].setPosition(Enemies[i].getPosition().x + (move), Enemies[i].getPosition().y);
217                     else if(Enemies[i].getPosition().x > 175 && Enemies[i].getPosition().x < 205 && Enemies[i].getPosition().y > 150
218                             && Enemies[i].getPosition().y < 360)
219                         Enemies[i].setPosition(Enemies[i].getPosition().x, (Enemies[i].getPosition().y - (move)));
220                     else if(Enemies[i].getPosition().x > 175 && Enemies[i].getPosition().x < 305 && Enemies[i].getPosition().y > 130
221                             && Enemies[i].getPosition().y < 200)
222                         Enemies[i].setPosition(Enemies[i].getPosition().x + move, (Enemies[i].getPosition().y));
223                     else if(Enemies[i].getPosition().x > 305 && Enemies[i].getPosition().x < 315 && Enemies[i].getPosition().y > 130
224                             && Enemies[i].getPosition().y < 480)
225                         Enemies[i].setPosition(Enemies[i].getPosition().x, (Enemies[i].getPosition().y + (move)));
226                     else if(Enemies[i].getPosition().x > 300 && Enemies[i].getPosition().x < 550 && Enemies[i].getPosition().y > 470
227                             && Enemies[i].getPosition().y < 500)
228                         Enemies[i].setPosition(Enemies[i].getPosition().x + move, (Enemies[i].getPosition().y ));
229                     else if(Enemies[i].getPosition().x > 540 && Enemies[i].getPosition().x < 560 && Enemies[i].getPosition().y > 260
230                             && Enemies[i].getPosition().y < 500)
231                         Enemies[i].setPosition(Enemies[i].getPosition().x, (Enemies[i].getPosition().y - move ));
232                     else if(Enemies[i].getPosition().x > 470 && Enemies[i].getPosition().x < 560 && Enemies[i].getPosition().y > 240
233                             && Enemies[i].getPosition().y < 270)
234                         Enemies[i].setPosition(Enemies[i].getPosition().x - move, (Enemies[i].getPosition().y ));
235                     else if(Enemies[i].getPosition().x > 450 && Enemies[i].getPosition().x < 480 && Enemies[i].getPosition().y > 95
236                             && Enemies[i].getPosition().y < 270)
237                         Enemies[i].setPosition(Enemies[i].getPosition().x, (Enemies[i].getPosition().y - move));
238                     else
239                         Enemies[i].setPosition(Enemies[i].getPosition().x + move, (Enemies[i].getPosition().y));
240                     //Above code moves the enemy along based on what point on the map it is
241                 }
242             }
```

Not only is this very unpleasing, it also isn't the most efficient way of doing it. In fact, as part of the problem specification, my game is supposed to increase in speed as it goes along, but this causes sufficient lag for the above code to not recognise the enemies as having gone off the path, and instead, the enemies begin to follow their own silly route which means the game then never ends.

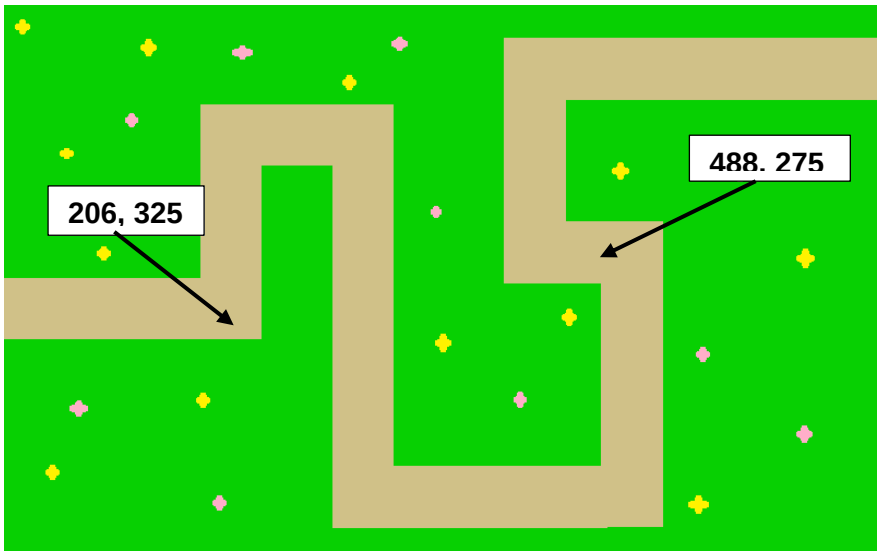
To fix this, I decided to use a different algorithm. Instead of constantly checking if enemies are falling within certain bounds and ranges, I could have them attract to certain points instead, so that once they are beyond this point, they can change direction. To do this, I first created an array of points on the map at which I want the enemies to change direction (and so follow the shape of the path):

```
25  attractionPoints[0][0] = 206; attractionPoints[0][1] = 325;
26  attractionPoints[1][0] = 206; attractionPoints[1][1] = 170;
27  attractionPoints[2][0] = 330; attractionPoints[2][1] = 170;
28  attractionPoints[3][0] = 330; attractionPoints[3][1] = 500;
29  attractionPoints[4][0] = 573; attractionPoints[4][1] = 500;
30  attractionPoints[5][0] = 573; attractionPoints[5][1] = 275;
31  attractionPoints[6][0] = 488; attractionPoints[6][1] = 275;
32  attractionPoints[7][0] = 488; attractionPoints[7][1] = 110;
33  attractionPoints[8][0] = 780; attractionPoints[8][1] = 110;
34
```

This is a 2D array that corresponds nicely to points on the map, and can be thought of like this:

X	Y
206	325
206	170
330	170
330	500
573	500
573	275
488	275
488	110
780	110

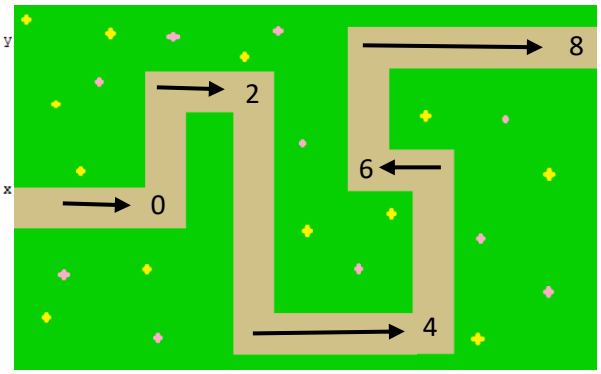
These points correspond to turning points on the map:



The difficulty lies in recognising which of the values (x or y) does the enemy need to cross for it to know that it has passed an attraction point. This can be done using a switch case:

```
71  bool attractMovement(int move, int x, int y)
72  {
73      switch(attractedNumber)
74      {
75          case 0: case 2: case 4: case 8:
76              sprite.move(move, 0);
77              if (sprite.getPosition().x + 20 > x)
78                  return true;
79              else
80                  return false;
81              break;
82          case 1: case 5: case 7:
83              sprite.move(0, -move);
84              if (sprite.getPosition().y + 20 < y)
85                  return true;
86              else
87                  return false;
88              break;
89          case 3:
90              sprite.move(0, move);
91              if (sprite.getPosition().y + 20 > y
92                  return true;
93              else
94                  return false;
95              break;
96          case 6:
97              sprite.move(-move, 0);
98              if (sprite.getPosition().x + 20 < x
99                  return true;
100             else
101                 return false;
102             break;
103     }
104 }
```

So if it is attraction point 0, 2, 4 or 8, the enemy needs to change direction given that it's x position is greater than the x position given by the attraction point, but for attraction point 6, it needs to have a x position less than the x value of the attraction point, as the enemy is travelling back on itself as can be seen on the map below:



A similar pattern exists for the y values. So now my code knows which value (either the x or y) it needs to look at and whether or not the y value of the enemy should be greater or less than this. The above code also ensures it travels in the right direction up to this point, which it does as if it needs to move left, the x value is negative otherwise it is positive and the same for up and down values of y (note that up for the y value is negative). From here, I can really simplify my update() function

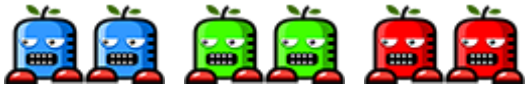
The code is much simpler and only needs to check if it is past its attraction point using the code noted earlier.

```
400 void update(sf::Time deltaTime)
401     {
402         float time = deltaTime.asSeconds();
403         int move = EnemySpeed * time;
404         //the enemy movement is speed * time to work out distance to be constant for all FPS
405
406         for (int i = 0; i < totalEnemies; i++)
407         {
408             if(Enemies[i].attractMovement(move, attractionPoints[Enemies[i].getAttractedNumber()][0],
409                                           attractionPoints[Enemies[i].getAttractedNumber()][1]))
410             {
411                 Enemies[i].incrementAttractionPoint();
412                 //If the enemy has passed the attraction point, make sure the enemy's attraction point has changed
413             }
414             if (Enemies[i].getAttractedNumber() == 9)
415             {
416                 towerHealth -= Enemies[i].getHealth();
417                 std::cout << towerHealth << std::endl;
418                 Enemies.erase(Enemies.begin() + i);
419                 totalEnemies--;
420                 i = 0;
421                 //If the enemy is at the last attraction point, it has reached the end of the map, and so, the tower's
422                 //health needs to decrement accordingly and the enemy removed from the screen.
423             }
424         }
425     }
```

If it is, then the next attraction point is used and if it is at the last attraction point, it attacks the tower and removes the enemy. This now works much smoother than the old code and is less prone to bugs due to lagging – it is successful.

The next update was largely an aesthetic feature. For the purposes of developing the game, my square enemies with different gradients were fine to use, but for the user, they would prefer enemies that actually looked nice. I also decided to implement some sort of animation for this and so I went from these:

To these:



With the first enemy being enemy 1 and so on. These are sprite sheets with two different textures – with each texture showing a different foot raised hence creating a sense of walking.

To use this, I simply loaded these textures instead and had a switchTime (currently set to 1 second) to switch the texture rect whenever the update time exceeded this switch time, much like with the powerup:

```
406         for (int i = 0; i < totalEnemies; i++)
407         {
408             if(enemyAnimationTimer >= switchTime)
409             {
410                 Enemies[i].nextAnimation();
411             }
```

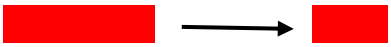
This time, the intrect was based on the remainder when the currentAnimation Integer was divided by 2. This is because I only have 2 textures for each enemy and so if the remainder is 0, I can use the first texture and if the remainder is 1, I can use the second texture (much like binary where 0 is off and 1 is on):

```
50      sf::Sprite getEnemySprite()
51      {
52          sprite.setTexture(texture);
53          sprite.setTextureRect(sf::IntRect(currentAnimation%2 * 40, 0, 40, 40));
54          return sprite;
55      }
```

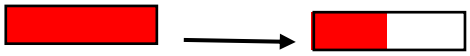
And so now my enemies looked better and were animated accordingly.

The next thing to fix was to implement health bars and reload bars. To do this, I used the sf::RectangleShape and created an object of this SFML class to form the outline and the inside colour. For the outline of both, I had a transparent rectangle with a black outline of thickness 1pixel. For the inside of the health bar, I used a solid red rectangle and for the reload bar I used a solid cyan rectangle.

To get the health bar to decrease in size according to the health of the enemy, I used percentages. For example, if the health of the enemy should be 50 but is 25, then the currentHealth/fullHealth gives us a percentage of 0.5 (50%). If we then set the size of the inside colour rectangle to be 0.5 * the size of a rectangle at full health, I can show the health bar to have decreased i.e. if the size of the rectangle at full health is 40pixelsx5pixels, then multiplying the x component by 0.5 will reduce the size to 20pixelsx5pixels and so do this:



To make this change noticeable, the transparent outline box comes into play:



And this now looks like a health bar. The code for this is as follows:

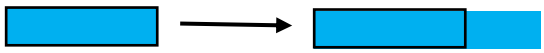
```
88          sf::RectangleShape getHealthBorder()
89          {
90              EnemyHealthBorder.setPosition(sprite.getPosition().x, sprite.getPosition().y + 50);
91              return EnemyHealthBorder;
92          }
93
94          sf::RectangleShape getHealthInside()
95          {
96              float multiple = EnemyHealth/TempHealth;
97              EnemyHealthInsideColour.setSize(sf::Vector2f((multiple) * 40, 5));
98              EnemyHealthInsideColour.setPosition(sprite.getPosition().x, sprite.getPosition().y + 50);
99              return EnemyHealthInsideColour;
100         }
```

So this does exactly what was explained above. As getHealthBorder() and getHealthInside() is called every time the window renders itself, this ensures that the health bar is updated every frame. In addition, the health border and health inside follows the enemy it corresponds to as it uses the enemy sprite position to be 50 pixels below the enemy at all times. A similar method was applied to the tower health bar, but the method for the reload bar is slightly different as shown on the next page:

```
147         sf::RectangleShape getReloadOutline()
148         {
149             if(TurretSprite.getRotation() > 90 && TurretSprite.getRotation() < 270)
150                 reloadOutline.setPosition(BaseSprite.getPosition().x, BaseSprite.getPosition().y - 10);
151             else
152                 reloadOutline.setPosition(BaseSprite.getPosition().x, BaseSprite.getPosition().y + 55);
153             return reloadOutline;
154         }
155         sf::RectangleShape getReloadInside()
156         {
157             if(TurretSprite.getRotation() > 90 && TurretSprite.getRotation() < 270)
158                 reloadInside.setPosition(BaseSprite.getPosition().x, BaseSprite.getPosition().y - 10);
159             else
160                 reloadInside.setPosition(BaseSprite.getPosition().x, BaseSprite.getPosition().y + 55);
161 
162             //If the turret is pointing downwards, the reload bar should be displayed at the top
163             //otherwise make it display at the bottom of the weapon
164 
165             if(currentTime.asSeconds()/reloadTime.asSeconds() >= 1)
166             {
167                 reloadInside.setSize(sf::Vector2f(40, 5));
168                 //If the current time exceeds the reload time, we don't want it to overshoot the bar
169             }
170             else
171             {
172                 reloadInside.setSize(sf::Vector2f(40 * currentTime.asSeconds()/reloadTime.asSeconds(), 5));
173                 //Normal percentages for the inside
174             }
175             return reloadInside;
176         }
```

Firstly, the position of the reload bar depends on which way the turret is facing. If the turret is facing down and the health bar is also below the weapon, then the turret may end up covering the health bar. This is bad and so to avoid this, the method shown above checks to see which way the turret is pointing. If it is facing upwards, the reload bar is displayed below the weapon but if it is facing downwards, the reload bar is displayed above the weapon.

In addition, this time, as the current time can exceed the reload time, I could have a problem whereby this happens:

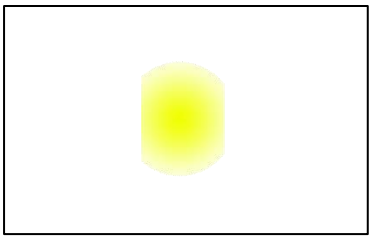


This is clearly wrong and is a problem. To avoid this, the above code also checks if the current time is greater than the reload time. If it is, it only fills the bar to its maximum and does not use percentages, and if it isn't it can safely use percentages.

Next is another aesthetic thing. The powerup animation is very displeasing in that it looks as though it is lagging. This needed some improvement. To do this, I decided to instead use the SFML sf::CircleShape class and create an object. To make the animation, I would instead create a function with respect to time whereby:

$$\frac{dr}{dt} = kt$$

So the rate of increase in the radius is equal to some constant multiplied by the time since the last update. Therefore, the radius will increase with time and will make the same animation the sprite sheet before was attempting (and uses a lot less RAM). However, in doing this, I lost the gradient I previously had and had to stick with using a solid circle instead:



This has a gradient and so looks a bit more like an explosion, but is very laggy and not realistic.

This has no gradient but works a lot smoother than the one before

```
518         void runPowerupAnimation(float time)
519         {
520             PowerUp.setPosition(400, 300);
521             powerupAnimationTimer += time;
522             PowerUp.changeRadius(powerupAnimationTimer);
523             //The radius is changed using the time since the last update
524
525             if(PowerUp.getAnimationRadius() >= 300)
526             {
527                 for(int i = 0; i < totalEnemies; i++)
528                 {
529                     Score = Score + Enemies[i].getScore();
530                     Money = Money + Enemies[i].getReturnMoney() * 5;
531                     Enemies.erase(Enemies.begin() + i);
532                     //Once the radius of the circle is big enough, it can
533                     //remove all the enemies
534                 }
535                 totalEnemies = 0;
536                 PowerUp.resetPosition();
537                 usingPowerUp = false;
538             }
539         }
```

The above function no longer needs to check which texture it is currently using, but instead check the radius of the circle. Once it exceeds 300, it has filled up the entire map and so the enemies can be removed and the money given to the user. This is much better.

Finally, back in Iteration 1, I forgot to have some sort of response to the user should they not be able to afford a weapon. This is important as otherwise they may not realise they cannot afford a weapon. To fix this I did two things:

1) If they did not have enough money to buy a weapon, the background on the weapon bar would be red. The same would be done for the powerup

To do this, I edited the method for drawing the weapon bar onto the screen, by sending over a parameter of money to the method. This enabled me to be able to compare the money they have to the cost, and if they had enough money, the background would be white, otherwise it would be red:

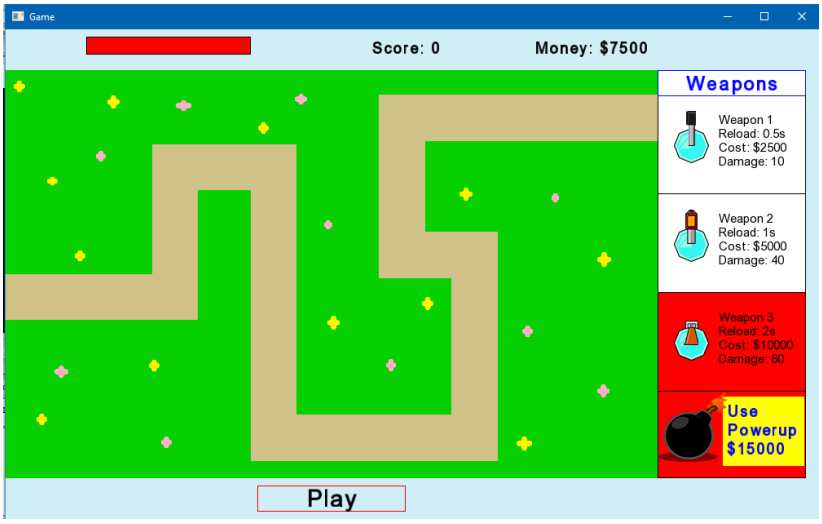
```
538         sf::RectangleShape getBorder(int money)
539         {
540             if(money < cost)
541             {
542                 Border.setFillColor(sf::Color::Red)
543                 //If they don't have enough money, the background goes red
544             }
545             else
546             {
547                 Border.setFillColor(sf::Color::White);
548                 //Otherwise it goes white
549             }
550             return Border;
551         }
```

2) If they did not have enough money, they would not be able to drag the weapon onto the map

Previously, even if they did not have enough money, they would be able to drag a weapon onto the map albeit the weapon would not get added onto the map. To avoid confusion, I made it so that they simply could not drag a weapon onto the map if they did not have the money. In order to do this, I edited the code for checking if a weapon is being dragged:

```
71  else if (Money - Weapon1.getCost() >= 0 && event.mouseButton.x > Weapon1.getBasePosition().x &&
72          event.mouseButton.x < Weapon1.getBasePosition().x + 42.5f &&
73          event.mouseButton.y > Weapon1.getBasePosition().y &&
74          event.mouseButton.y < Weapon1.getTurretPosition().y + 64.25f)
75  {
76      isDragging1 = true;
77  }
```

One of the new additional criteria is that the user must have enough money to buy the weapon for the code to register a drag, otherwise it will not drag. This works as shown to the right. The user cannot afford Weapon 3 or the PowerUp.

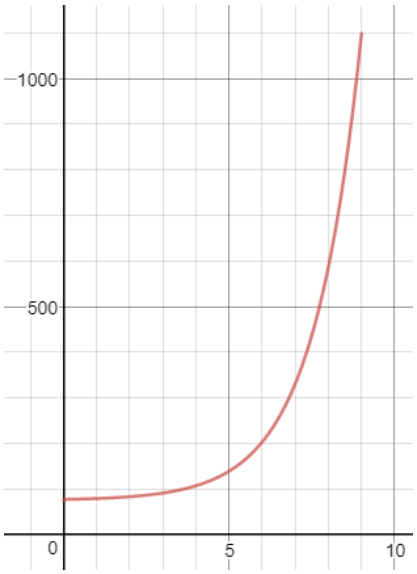


Also, the balance of the game was key as the entire problem of current tower defence games was that they were largely unbalanced. To balance the game, I had the following features:

1) The money they have at the start is $7500. This is enough to buy a Weapon 1 and Weapon 2 or Three Weapon 1's and will just get them through to the next wave.
2) The initial score is set to 100. This produces enemies which are just tough enough to cause the user grief but not tough enough to make the game impossible.
3) The enemies that come are based on the user's score as explained back in Iteration 3. This ensures that the user is deep enough in the game before experiencing and real problems.
4) The enemy speed exponentially increases based upon the user's score. This also follows the updated Problem Specification and is sufficient enough to cause the user grief in later stages and force them into using strategies to help them defeat a wave of enemies. This seemed to be a perfect value:

```
EnemySpeed = Score/50 + 75;
```

This causes the following increase in difficulty/speed per wave of enemy:



As can be seen, by around the 9th wave, the difficulty has increased 11 fold even though by the 5th wave, the difficulty has only increased about 2 fold.

The money received for every enemy killed is also just enough to allow them to buy weapons to get them through the next wave, given that they deploy a reasonable strategy. This is because it is largely based on the user's score, and they can expect to get about twice as much money for every level given that they manage to kill all the enemies. This creates a very balanced game.

To increase the strategy side, I also have Weapon 2 as being more cost effective than Weapon 3, and so the user needs to realise this to ensure an easier victory. This should help with the strategy side of the game.

The game (now half finished) was given to my 6 stakeholders for feedback. They were allowed to play the game for around 15 minutes each and while they were playing, the reported 3 things

1) Any bugs they noticed in the game – This would help me ensure that I haven't missed any bugs which the user would notice but a developer wouldn't
2) Any additions they think would make the game better – This would ensure that the problem specification I wrote earlier on was correct and I am following it
3) General Comments about the game (is it good) – Just to understand if my success criteria is being fulfilled so far.

These were the responses:

Interview Session 2

Having completed Iteration 6 of the Development, the game was now playable and hence I was able to interview my stakeholders on the current progress of the game. This involved them playing the game for around 15 minutes and reporting back to me with grievances and what they liked as well as answering my questions.

Praveen Murugathas – Interviewed on 23/11/2017 at 8:35AM

Feedback received from Praveen:

Bugs:

1. You cannot seem to add weapons near the start of the path on the bottom part. This clearly should not be so and significantly reduces the number of places weapons can be added. Therefore, this should be fixed.
2. Some reload bars don't seem to have a black outline near the right of them (this may have been a one-off but test this nevertheless)

Recommended Additions:

1. Pop-up Screen at end showing money spent, money earned and score
2. This pop-up screen should have your health, your score and the money spent as well as maybe "You can afford…."

General Feedback on the game:

1. The game is very fun and addictive. I didn't want to stop playing (this was evident as Praveen was very reluctant to move onto the questions stage and instead wanted to continue playing)
2. The game is aesthetically pleasing – the enemies and weapons look nicer than was expected
3. A black outline should be added to the outside of the path and the flowers just so that they stand out a bit more.
4. The path should be a bit more curved to look realistic

Michael Kuc – Interviewed on 23/11/2017 at 11:20AM

Feedback received from Michael:

Bugs:

1. Locking onto enemies is not prioritised
2. Placing turrets in white areas doesn't always correspond to it being added. Why?

Recommended Additions:

1. Background should have a little more texture to it.
2. Weapon 3 should be worth it, it shouldn't be that Weapon 2 is better than Weapon 3
3. Health bars shouldn't switch as often as they do and neither should enemy animations
4. Weapons should be able to be moved once placed
5. Like the flowers
6. Difficulty increase should be less but then increase in money also has to be less
7. Quadratic increase in difficulty not exponential
8. Have a function of towers and current money to work out the score you get at the end.

General Feedback on the game:

1. Liked the flowers
2. Health bars are brilliant
3. Really like reload time bars

Miron Abhaysinghe – Interviewed on 23/11/2017 at 1:50PM

Feedback received from Miron:

Bugs:

1. The priority for choosing which enemy to kill is all wrong. Fix it.

Recommended Additions:

1. Improve the map so that It looks better
2. All weapons should have a larger radius

General Feedback on the game:

1. It is a fun and addictive game
2. The prices are sometimes a bit high so maybe fix it
3. I would love this game after all the improvements are made

Nathan Wang – Interviewed on 23/11/2017 at 2:10PM

Feedback received from Nathan:

Bugs:

1. The prioritising of weapons is all wrong. They don't lock on to the first enemy, only the last.
2. There is a bug where you can actually place the weapon onto the path in some cases
3. There is a bug in terms of layering when it comes to the weapon menu (the weapon hides behind the powerup section)

Recommended Additions:

1. If possible, have the ability to upgrade weapons
2. Be able to remove weapons and move weapons around
3. Improve weapon 3 to be a bit more worth it then it currently is

General Feedback on the game:

1. The game is "pretty good"

Vinayak Shastri – Interviewed on 24/11/2017 at 8:25AM

Feedback received from Vinayak:

Bugs:

1. Cannot add weapons along the bottom even though a white radius is shown
2. Can place weapons on a small part of the track
3. The prioritising of weapons locking on is all wrong

Recommended Additions:

1. Should be able to move weapons once they have been placed
2. Improve Weapon 3 to be more worth it
3. Make it more obvious that the powerup button is a button

General Feedback on the game:

1. It is a very addictive game
2. Extremely fun

Oliver Wales – Interviewed on 24/11/2017 at 8:45AM

Feedback received from Oliver:

Bugs:

1. The tower health bar actually goes beyond 0. Add a limit to it
2. The priority of weapons needs fixing. It's all wrong
3. Fix the weapon placing boundary to be more consistent

Recommended Additions:

1. Weapon 3 should have a greater radius

General Feedback on the game:

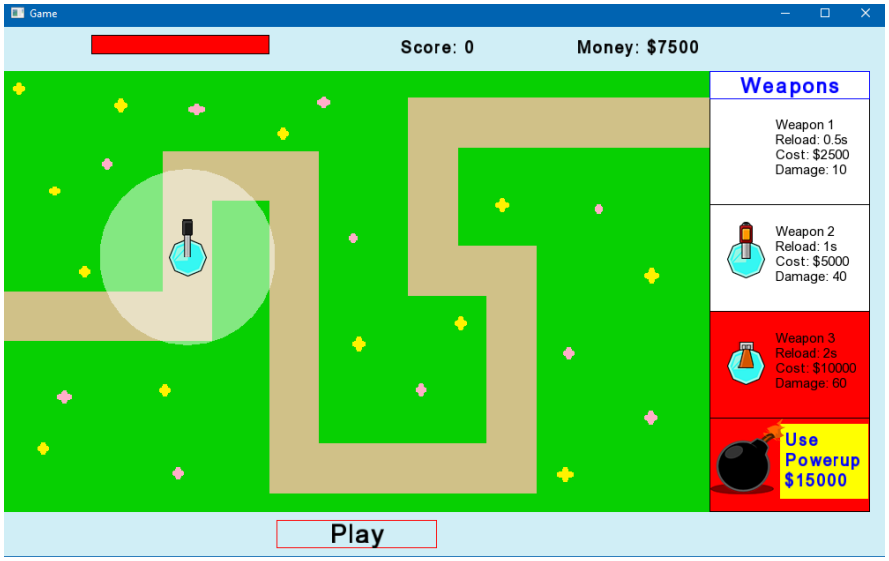
1. "It's a stupid game"
2. "I don't really like it. Personally I'd rather play Bloons TD4 which was released in 2006"

Oliver Wales clearly did not seem to be impressed by the game. However, my other 5 stakeholders were very happy with the game and found it to be fun and addictive and therefore we can assume that Oliver was an anomaly and very different from the rest. Therefore, his general feedback can be disregarded as a rare occurrence.

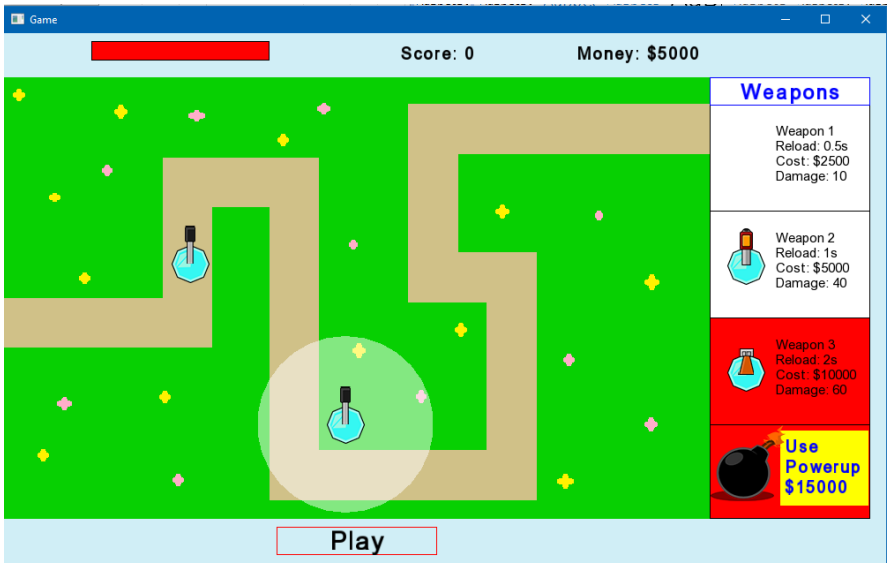
Overall feedback and improvements I need to make:

The first thing that I need to address is the request for the ability to move weapons that have already been placed. This was requested by 3 of my stakeholders but is not something that needs immediate attention, as it is already scheduled to be included as part of Iteration 7.

Next almost all of my stakeholders noticed bugs with placing weapons. This clearly needs to be addressed but seems to be an easy fix. The biggest bug comes in a place where you can actually place a weapon on the path, which is clearly a major issue:



While other weapon placing bugs include a lack of consistency in how close weapons can be placed from the path:



The layering bug is an easy fix and although it was only noticed by one, I will fix this.

The reload bar bug mentioned by Praveen was not noticed by me or any of the other 5 stakeholders and may have been a problem of the device it was loaded on.

However, weapon lock priority was a major issue raised by 5 of my stakeholders, and so will need to be addressed. In addition, 4 of my stakeholders said Weapon 3 needs to be more worth it while 1 stakeholder said that all weapons should have a larger radius. To address this issue, I will make Weapon 3 have a larger radius.

I will also address Vinayak's issue of the powerup button not being obvious and Oliver's issue of the tower health bar not having a minimum value.

To address the issue of consistency in weapon placement, I edited the mouse move event checker so that the pixels it checks correspond to the path only (as in it only checks if the user is attempting to place a weapon on the path) and also to check if the weapon is being placed off the map rather than 50 pixels in from the map:

```
;
if (event.mouseMove.x > 0 && event.mouseMove.x < 257.25 && event.mouseMove.y > 278.25 && event.mouseMove.y < 376.25)
    Radius.setFillColor(sf::Color(242,43,21,128));
else if (event.mouseMove.x < 257.25 && event.mouseMove.x > 157.75 && event.mouseMove.y < 333.75 && event.mouseMove.y > 118.75)
    Radius.setFillColor(sf::Color(242,43,21,128));
else if (event.mouseMove.x < 378.25 && event.mouseMove.x > 157.75 && event.mouseMove.y < 217.25 && event.mouseMove.y > 118.75)
    Radius.setFillColor(sf::Color(242,43,21,128));
else if (event.mouseMove.x < 378.25 && event.mouseMove.x > 278.75   && event.mouseMove.y > 118.75)
    Radius.setFillColor(sf::Color(242,43,21,128));
else if (event.mouseMove.x < 624.25 && event.mouseMove.x > 278.25 && event.mouseMove.y > 449.75 && event.mouseMove.y < 550)
    Radius.setFillColor(sf::Color(242,43,21,128));
else if (event.mouseMove.x < 624.25 && event.mouseMove.x > 524.75 && event.mouseMove.y > 225.75 && event.mouseMove.y < 550)
    Radius.setFillColor(sf::Color(242,43,21,128));
else if (event.mouseMove.x < 624.25 && event.mouseMove.x > 435.75 && event.mouseMove.y > 225.75 && event.mouseMove.y < 325.25)
    Radius.setFillColor(sf::Color(242,43,21,128));
else if (event.mouseMove.x < 536.25 && event.mouseMove.x > 435.75 && event.mouseMove.y < 325.25)
    Radius.setFillColor(sf::Color(242,43,21,128));
else if (event.mouseMove.x > 435.75 && event.mouseMove.y < 157.25)
    Radius.setFillColor(sf::Color(242,43,21,128));
else if (event.mouseMove.x < 21.25 || event.mouseMove.x > 778.75 || event.mouseMove.y < 71.25 || event.mouseMove.y > 528.75)
    Radius.setFillColor(sf::Color(242,43,21,128));
else if(!toggle)
    Radius.setFillColor(sf::Color(255,255,255,128));
```

In addition, to address the issue of consistency in placing weapons near each other, I made the x and y values for the checker the same as can be seen below:

```
11
12
13
14
15
16
17
18
19
20
21
22
23
```

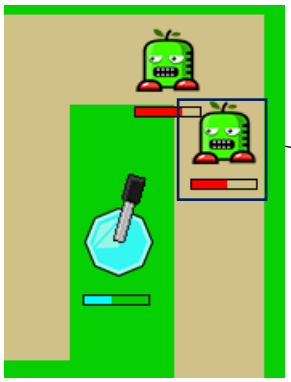
```
for (int i = 0; i < totalWeapons; i++)
{
    if (event.mouseMove.x < Weapons[i].getBasePosition().x + 63.75
        && event.mouseMove.x > Weapons[i].getBasePosition().x  - 21.25
        && event.mouseMove.y < Weapons[i].getBasePosition().y + 63.75
        && event.mouseMove.y > Weapons[i].getBasePosition().y - 21.25)

    {
        Radius.setFillColor(sf::Color(242,43,21,128));
        toggle = true;
        break;
    }
}
```

As a result, I no longer have weapon placement bugs. For the layering issue, this came down to my rendering function rendering the weapons before the borders and to fix this, I simply reordered it to render the weapon after the border:

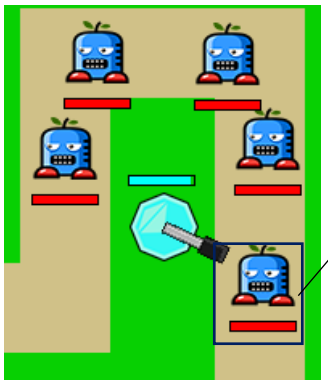
```
mWindow.draw(Weapon2Text);
mWindow.draw(Weapon3Text);
mWindow.draw(WeaponsTitleBorder);
mWindow.draw(WeaponsTitle);
mWindow.draw(PowerUp.getBorder(Money));
mWindow.draw(PowerUp.getButton());
mWindow.draw(PowerupButtonText);
if (isDragging1 || isDragging2 || isDragging3)
{
    mWindow.draw(Radius);
}
mWindow.draw(Weapon1.getBaseSprite());
mWindow.draw(Weapon1.getTurretSprite());
mWindow.draw(Weapon2.getBaseSprite());
mWindow.draw(Weapon2.getTurretSprite());
mWindow.draw(Weapon3.getBaseSprite());
mWindow.draw(Weapon3.getTurretSprite());
```

And finally, I increased Weapon 3's radius as requested. This did not require major code editing as all I needed to do was change a value in the Weapon3Class to have a radius greater than both Weapon1 and Weapon2. The difference in radii can be seen below:

The weapon priority came down to a very small error on my part. When choosing which enemy to lock onto, I forgot to put in a break; once it had locked on to an enemy. This meant that it kept checking for enemies to lock onto and instead of locking on to the first enemy in its radius and staying locked on, it locked onto the last enemy. By putting in a break; the code would no longer have this problem and would always lock onto the first enemy to enter its radius:



Before: Should be locked on this enemy



After: Should be locked on this enemy

So this problem was fairly easy to fix:

```
for (int j = 0; j < totalEnemies; j++)
{
    if(isInRange(Weapons[i].getBasePosition(), Enemies[j].getPosition(), Weapons[i].getRadius()))
    {
        Weapons[i].lockOnto(j);
        break;
        /* Debugged. Locked onto all but continued loop. Put in break case
        if (j == 0)
        {
            std::cout << "Locked onto enemy " << j << std::endl;
        }*/
    }
}
```

To limit the Tower Health Bar I inserted the following code (similar to the one I had previously used for the reload bars):

```
float percentage = towerHealth/fullHealth;
if (percentage <= 0)
    TowerHealthInside.setSize(sf::Vector2f(0, 20));
else
    TowerHealthInside.setSize(sf::Vector2f(percentage * 200, 20));
mWindow.draw(TowerHealthOutline);
```

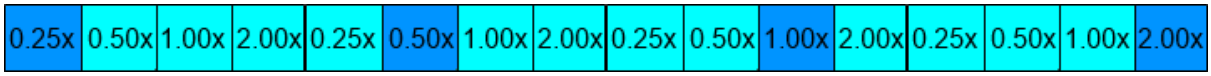
So now if the health was below 0, it would have a predefined size it could not go below.

For the powerup button, I simply added the text “Click Button” to make it more obvious that the button should be used to use the powerup:

And so I had addressed the most important user feedback I had got and fixed the bugs. Iteration 6 was now complete and I could move on to Iteration 7.

Iteration 7 – Speed Up, Speed Down Modules, Buton Pressed Module

One part of the problem specification was that users should be able to increase and decrease the speed of the animation by pressing buttons. Therefore, the next step was to do exactly that. To do this, I created a sprite sheet for button selection:



The dark blue boxes represent the current speed that has been selected.

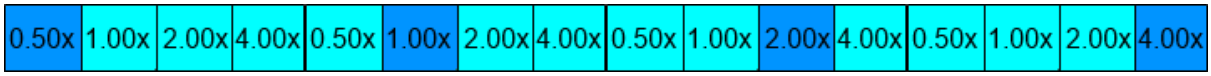
Next, I edited my code so that these would function properly. The code simply uses the intrect function defined before so that whenever the user clicks on a certain button, the sprite sheet rectangle updates so that a different button can be shown to have been clicked. To ensure the speeds are changed, I had a changeEnemySpeed function:

```
331     void changeEnemySpeed()
332     {
333         EnemySpeed = EnemySpeed/tempSpeedMultiplier;
334         EnemySpeed = EnemySpeed * speedMultiplier;
335         tempSpeedMultiplier = speedMultiplier;
336     }
```

tempSpeed stored the speed that the user had before and dividing EnemySpeed by this gets the enemy speed back to normal. Then multiplying by the new speedMutliplier allows the speed to chane to the user requirements. To change the weapon reload times, I simply had the time sent to the weapon class to be multiplied

by the speedMultiplyer so that if 1 second had passed but the speed mutiplier was 0.5, the weapon would assume only 0.5seconds had passed etc.

The problem lies with 0.25x speed. This causes the enemy to simply stop moving at early stages in the game. The reason for this comes down to they way floats work. As we are now working with decimal numbers, 0.005 will still be deemed a valid number by my code. However, a CPU may see this as so insignificant that the code does not move the enemy along at all and instead treats 0.005 like 0. This only happens when the value of the speed multiplier is very low, and therefore, it was better to simply reject 0.25x speed as an option and have the following sprite sheet instead:



Tests)

	Problem Being Tested	Action to be taken to test this problem	Expected Result
18	Iteration 7 - Speed	Click 0.5 speed button	Speed decreases
19		Click 2.0 speed button	Speed increases
20		Click 5.0 speed button	Speed increases

As explained, the speed options were changed, but they do fully work (but it is hard to show it working through screenshots). This concluded Iteration 7.

Iteration 8 – Weapon Movement, Sell Weapon Module, Load Help Screen, Pause Screen and Game End Screen Modules

To move weapons, I had to re-edit the code for processPreGameEvents() to recognise the user clicking on a weapon.

```
for (int i = 0; i < totalWeapons; i++)
{
    if (event.mouseButton.x > Weapons[i].getBasePosition().x
        && event.mouseButton.x < Weapons[i].getBasePosition().x + 42.5
        && event.mouseButton.y > Weapons[i].getBasePosition().y
        && event.mouseButton.y < Weapons[i].getBasePosition().y + 42.5)
    {
        //If clicked on a placed weapon, then change the bool value
        selectedPlacedWeapon = true;
        selectedPlacedWeaponNumber = i;
        //Store the number of the weapon that has been clicked on
        Radius.setRadius(Weapons[selectedPlacedWeaponNumber].getRadius());
        preBasePosition = Weapons[selectedPlacedWeaponNumber].getBasePosition();
        preTurretPosition = Weapons[selectedPlacedWeaponNumber].getTurretPosition();
        //And store its current position
        break;
    }
    else
    {
        selectedPlacedWeapon = false;
    }
}
```

The code checks if the user has clicked on a placed weapon. If it has, it stores the weapon id, the current position and changes the bool value of selected Placed Weapon to true.

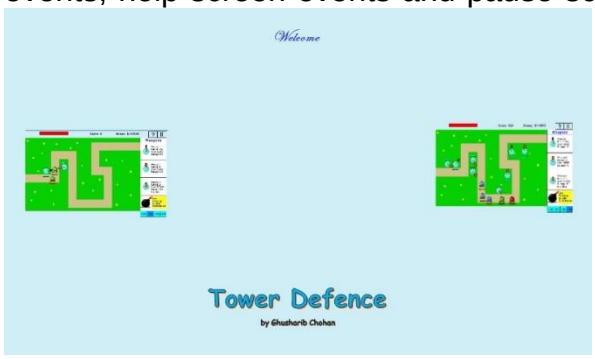
It can then check using the previously defined values if the weapon has been placed in a valid position. If it has, it does nothing (as it has moved to this valid position already) but if it hasn't, then it places the weapon back in its original position:

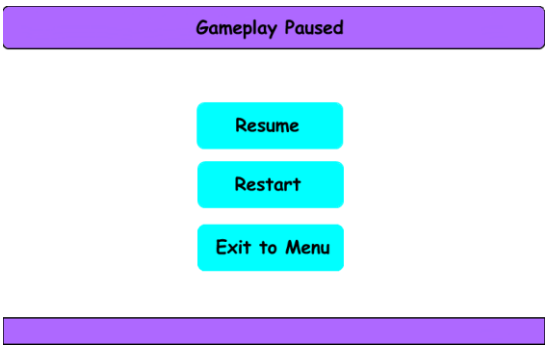
```
if (placedIncorrectly && selectedPlacedWeapon)
{
    std::cout << "Called" << std::endl;
    Weapons[selectedPlacedWeaponNumber].setBasePosition(preBasePosition.x, preBasePosition.y);
    Weapons[selectedPlacedWeaponNumber].setPostPlacedTurretPosition(preTurretPosition.x, preTurretPosition.y);
    selectedPlacedWeapon = false;
}
```

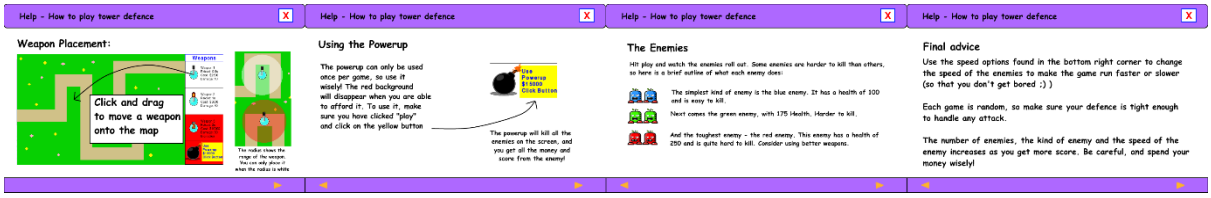
If the weapon is dragged to the part of the screen where the weapon can be sold, the weapon is sold and money is returned to the user (this is half the cost of buying the weapon to avoid user's abusing the sell ability).

```
if(event.mouseButton.x > sellWeaponButton.getPosition().x && event.mouseButton.x < sellWeaponButton.getPosition().x + 180
   && event.mouseButton.y > sellWeaponButton.getPosition().y && event.mouseButton.y < sellWeaponButton.getPosition().y + 40)
{
    Money = Money + Weapons[selectedPlacedWeaponNumber].getReturnMoney();
    Weapons.erase(Weapons.begin() + selectedPlacedWeaponNumber);
    totalWeapons--;
    selectedPlacedWeapon = false;
}
```

And so that concludes the weapon moving ability and selling the weapon. The rest of the iteration is just producing screens and buttons that work. To do this, I have separate functions for processing pre-game events, during game events, main menu events, help screen events and pause screen events. The screens were as follows:







The buttons were created exactly like the previous buttons in previous iterations. The main code here is the void run() function:


```
void run()
{
    sf::Clock clock;
    while (mWindow.isOpen())
    {
        if (showMainMenu)
        {
            if(!loadHelpScreen && !gamePaused)
            {
                processMainMenuEvents();
            }
            mainMenuRender();
        }
        else if(!playingGame)
        {
            if(!loadHelpScreen && !gamePaused)
            {
                processPreGameEvents();
            }
            clock.restart();
            BulletTimer.restart();
            preGameRender();
        }
        else
        {
            if(!GameOver)
            {
                if (clock.getElapsedTime().asSeconds() > TimePerFrame && !loadHelpScreen && !gamePaused)
                {
                    processDuringGameEvents(BulletTimer.getElapsedTime());
                    update(clock.getElapsedTime());
                    if(usingPowerUp)
                    {
                        runPowerupAnimation(clock.getElapsedTime().asSeconds());
                    }
                    clock.restart();
                }
                else if(clock.getElapsedTime().asSeconds() > TimePerFrame)
                {
                    clock.restart();
                }
                BulletTimer.restart();
            duringGameRender();
            }
            else
            {
                processGameOverEvents();
                gameOverRender();
            }
        }

        if(loadHelpScreen)
        {
            processHelpScreenEvents();
            helpScreenRender();
        }

        if(gamePaused)
        {
            processPauseScreenEvents();
            pauseScreenRender();
        }

        mWindow.display();
    }
}
```

This function simply ensures that if the help screen or pause screen are on, then game events are not being processed and so the game play pauses at that point. In addition, it ensures that any main menu or in game buttons are no longer pressable.

Tests for Iteration 8

1	Problem Being Tested	Action to be taken to test this problem	Expected Result
21	Iteration 8 Menus and buttons and help screen	Help button pressed during simulation	Help screen appears
22		Automatic	Next wave of attackers appears on bottom
23		Pause button pressed	Gameplay is paused
24		User navigates away from Game Window	Gameplay is paused
25		Click on weapon and click sell weapon	Weapon is removed and money increased
26		Help button pressed from main menu	Help screen appears
27		Start New Game button pressed from main menu	Game begins

Again, it is hard to show that these functions work, but they were tested and they work 100% fine as required. Therefore, this Iteration is now complete.

Iteration 9 – Save High Score and Load High Scores Modules

This is perhaps the most complex part of my game – using a high scores file and reading/writing to it with highscores placed in the correct position within the file. To do this, I created a vector that was populated with all the items in the high score file using a scores class I created. The scores class contains the position number, username and score of the user. This was the code for populating the vector:

```
600         void loadScores()
601         {
602             Scores.clear();
603             totalScores = 0;
604             //Clear and reset the vector
605             std::ifstream reader("Save File/HighScores.txt");
606             //Open the file
607             ScoresClass newScore;
608             //Create a new class
609             while (!reader.eof())
610             {
611                 ScoresClass newScore;
612                 std::string line;
613                 int x = 0;
614                 getline(reader, line, ',');
615                 std::stringstream line1;
616                 line1.str(line);
617                 line1 >> x;
618                 newScore.setPosition(x);
619
620                 std::string name;
621                 getline(reader, name, ',');
622                 newScore.setName(name);
623
624                 getline(reader, line, '\n');
625                 std::stringstream line2;
626                 line2.str(line);
627                 line2 >> x;
628                 newScore.setScore(x);
629
630                 Scores.push_back(newScore);
631                 totalScores++;
632                 //For every item in the file and while not at the end of the file,
633                 //add the item into the vector
634             }
635
636             reader.close(); //Close the file
```

The code iterates through each line of the file, where values in the file are seperated by commas in a comma seperated values file. It converts strings which should be integers into integer values using the string stream library and puts them into the vector as such. With each iteration, the counter for total scores is incremented and once the end of the file has been reached, the file is closed to avoid corruption.

I decided to use a binary search to find where in the file the new score should be added. This code is shown below:

```
void saveScore()
{
    int upperBound = totalScores, lowerBound = 0, middleValue = 0;
    bool found = false;
    int position;

    while (upperBound != lowerBound)
    {
        middleValue = (upperBound + lowerBound)/2;
        int comparison = Scores[middleValue].getScore();
        if (comparison == Score)
        {
            break;
        }
        else if(comparison < Score)
        {
            lowerBound = middleValue;
            middleValue--;
        }
        else
        {
            upperBound = middleValue;
            middleValue++;
        }
        std::cout << totalScores << " " << lowerBound << " " << comparison << " " << upperBound << std::endl;
    }

    std::cout << "The score of " << Score <<  " comes in position " << middleValue << std::endl;
}
```

This doesn't work. This comes down to the fact that Binary Search is made to find a

value that already exists within a sorted list, but it's likely that the score of the user is unique and it is not already in the file. It therefore instead returns a position value which is not the position the value should be placed in, but instead a nearby position. This can be fixed, but it means that this is not a true binary search, but instead an adapted version for my specific problem. Another problem of this current algorithm is that it fails to recognise when a user might have got a score exactly the same as a score which already exists, and forces the position of the already existing value to be greater than it should be i.e. if two people score 850, they should both have a position of e.g. 7 but instead it causes one to have a position 7 and another position 8.

The following algorithm fixed this problem:

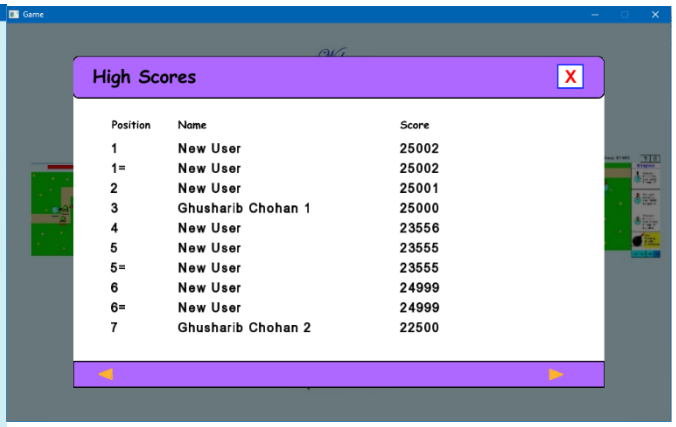
```
void saveScore(std::string name)
{
    int upperBound = 0, lowerBound = totalScores, middleValue = 0;
    bool breakNow = false;
    int position;
    //Sets variables for the binary search

    while (!breakNow)
    {
        middleValue = (upperBound + lowerBound)/2;
        int comparison = Scores[middleValue].getScore();
        if (comparison == Score)
        {
            break;
        }
        else if(comparison < Score)
        {
            lowerBound = middleValue;
        }
        else
        {
            upperBound = middleValue;
        }

        if(upperBound == lowerBound || upperBound - 1 == lowerBound || lowerBound - 1 == upperBound)
            breakNow = true;
    }
    //Finds the nearest value for the position using the binary search

    if(Scores[middleValue].getScore() == Score)
    {
        position = Scores[middleValue].getPosition();
        //If the score is equal to the score of the binary search value found, then we set its position as being the same
        //to avoid it causing the problems experienced before
    }
        //Then the position will be greater
    }
    while (Scores[middleValue].getScore() > Score)
    {
        position = Scores[middleValue].getPosition() + 1;
        middleValue = middleValue + 1;
        //The while loop ensures that the value doesn't fall in the middle of repeated values
    }
    position--;
}
else if (Scores[middleValue].getScore() < Score)
{
    if(Scores[0].getScore()  < Score || Scores[0].getScore() == Score)
    {
        position = 1;
        middleValue = 0;
        //If the score is less ten the position will be smaller
    }
    else
    {
        while (Scores[middleValue].getScore() < Score)
        {
            position = Scores[middleValue].getPosition() - 1;
            middleValue = middleValue - 1;
            //The while loop ensures that the value doesn't fall in the middle of repeated values
        }
        middleValue++;
    }
```

As explained in the comments, this algorithm now factors in repeat values and the fact that the binary search only gets a nearby position and not an accurate position. This therefore now works as required. The ability to type in your own name and a main menu high scores button was also implemented:

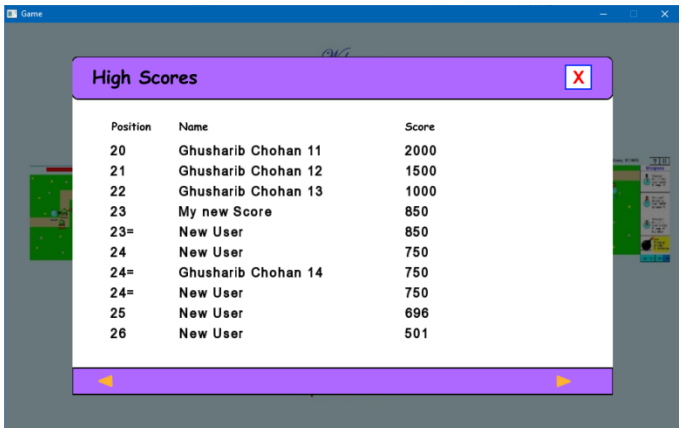
```
    ScoresClass newScore;
    newScore.setPosition(position);
    newScore.setName(name);
    newScore.setScore(Score);
    Scores.push_back(Scores[totalScores - 1]);
    //This creates a new object to place into the vector

    for (int i = totalScores - 1; i > middleValue; i--)
    {
        Scores[i] = Scores[i-1];
        //This moves all the relavant objects up in the vector
        //as in all the values after the new value
    }

    Scores[middleValue] = newScore;
    totalScores++;

    if (Score != Scores[middleValue + 1].getScore() && Score != Scores[middleValue - 1].getScore())
    {
        for(middleValue + 1; middleValue < totalScores + 1; middleValue++)
        {
            Scores[middleValue].incrementPosition();
            //Increment the position of all values after the value being added
        }

        if(position == 1)
        {
            Scores[0].setPosition(1);
            //If it's first, then set it's position to 1
        }
    }
    else
    {
        if(Score == Scores[position - 1].getScore())
        {
            Scores[position].setPosition(Scores[position - 1].getPosition());
        }
        else if(Score == Scores[position + 1].getScore())
        {
            Scores[position].setPosition(Scores[position + 1].getPosition());
        }
        //Find the correct position
    }


    std::ofstream writer("Save File/HighScores.txt");

    for(int i = 0; i < totalScores; i++)
    {
        writer << Scores[i].getPosition();
        writer << ',';
        writer << Scores[i].getName();
        writer << ',';
        writer << Scores[i].getScore();

        if (i != totalScores - 1)
        {
            writer << '\n';
        }
        //For every item in the vector, add it into the new highscores file to be saved
    }

    writer.close(); //Close the file
}
```

Tests for Iteration 9:

1) High Scores button. I click on the high scores button from the main menu – highscores are shown in the correct order.



2) End of the game, have the ability to save score



This concludes Iteration 9. Iteration 10 was cancelled as it was unecessary and there was a pressure on time in terms of this coursework. Therefore, the final game has now been produced.

In terms of graphics, all of the graphics used in the game are either produced by me or using a royalty free source as mentioned below:

[https://www.gamedevmarket.net/, Accessed in 2017]

Post-Development Testing

I am now able to test the final game myself using the black box test plan I made back in the design stage. A reminder of this test plan can be found as part of my Appendixes.

Appendix D shows the test plan completed while the following pages shows the detail of me carrying out the tests.

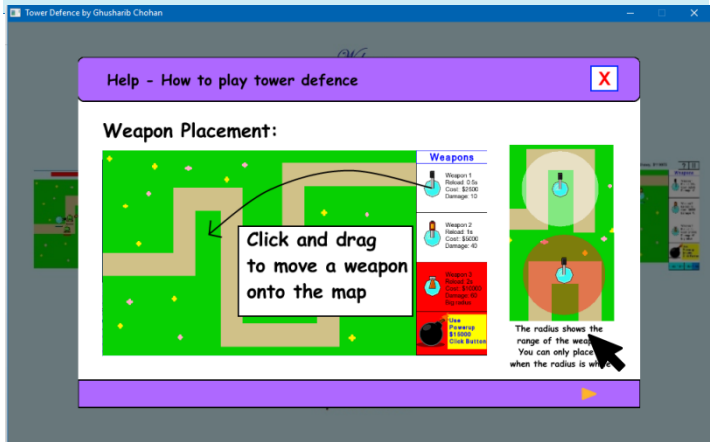
Menu:

Test 1) Click on Help Button (These follow the Help Button Testing Section):

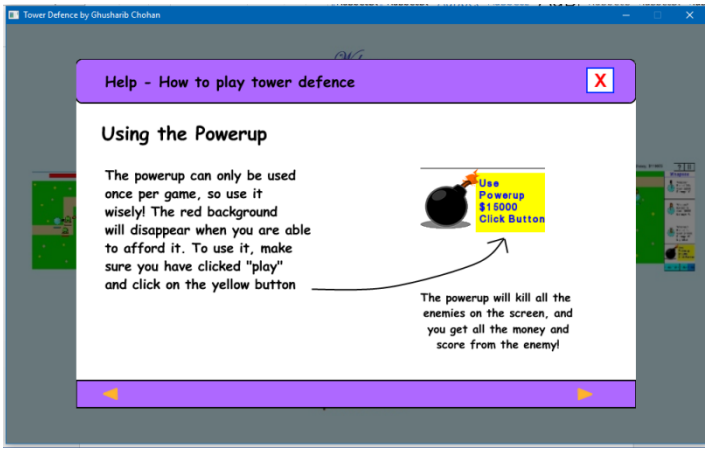
Help Button:

Test 1a) Click the help button:

I click on the “Help” button from the main menu

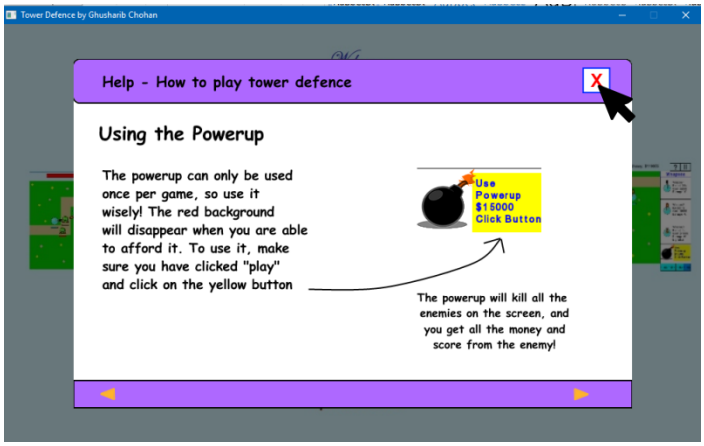


The help screen loads and I press on the “Next” Button

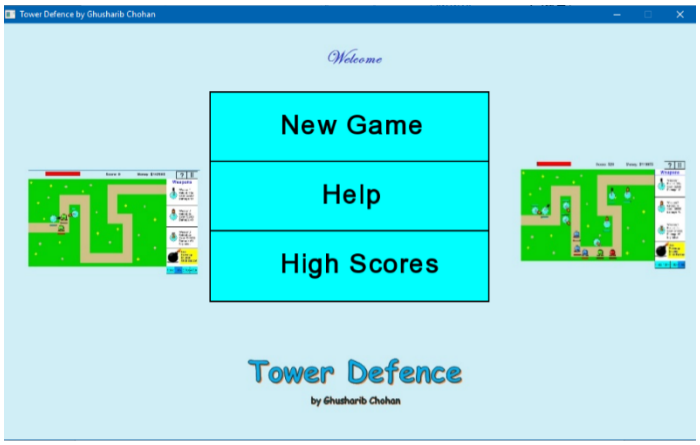


The help screen displays and pressing on the next button takes me to the next help screen and so on **[TEST PASSED]**

Test 1b) Click on close button on the help screen



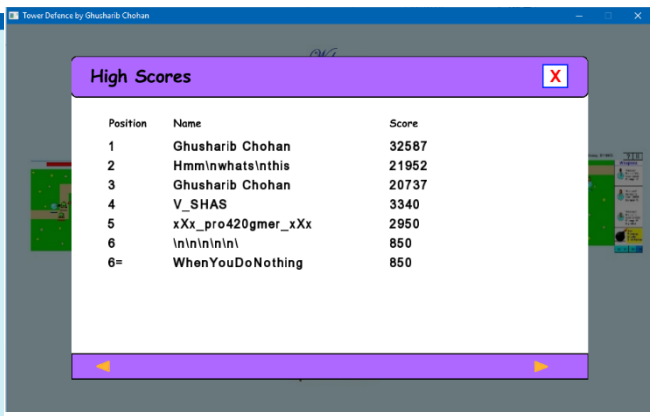
I click on the “Close” Button on the Help Screen.



Help screen closes and returns to the previous screen **[TEST PASSED]**

Continue Menu:

Test 2) Click on High Scores Button:

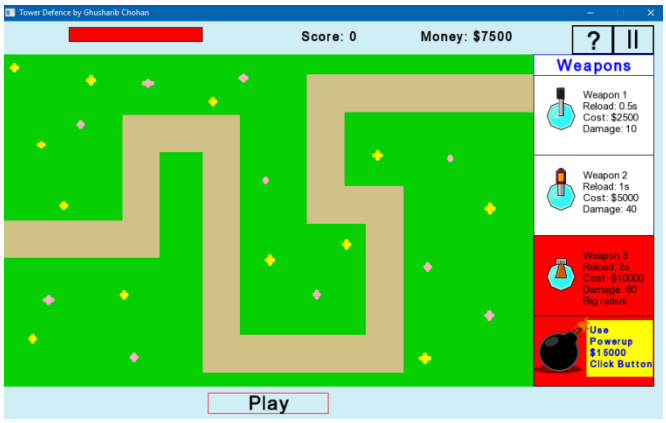


High Scores are displayed and appear in the correct order (with descending scores)

[TEST PASSED]

Test 3) Click on New Game

I click on the “New Game” button from the Main Menu

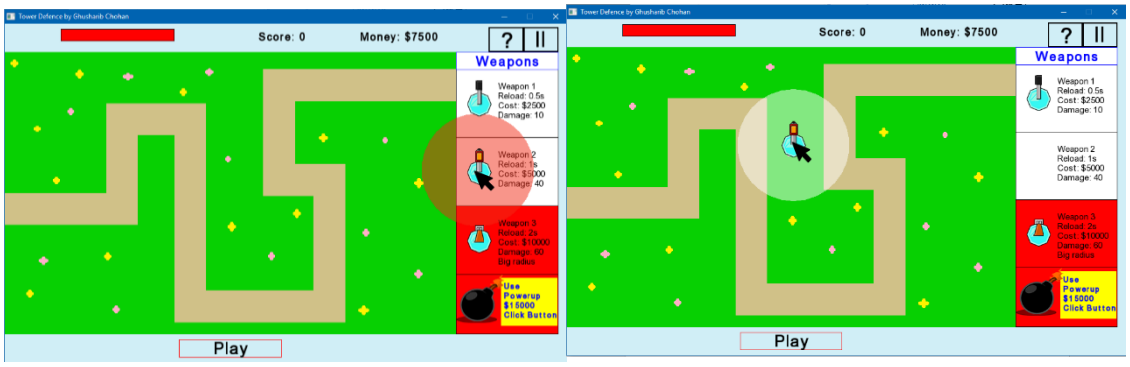


New Game starts with 0 Score and $7500 in money (to allow them to buy weapons) with a blank map.

[TEST PASSED]

Click and Drag Feature

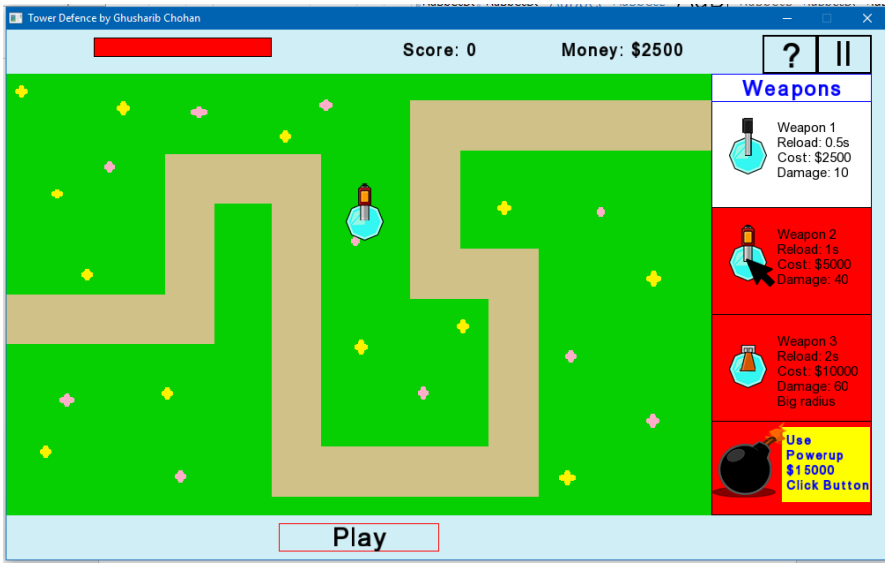
Test 1) Add Weapon with enough money and to a valid place



The Weapon has been added to the part of the screen it was dragged to and the money has decreased **[TEST PASSED]**

Test 2) Click on weapon and attempt to drag to a valid part of screen without enough money

Test 3) Click on weapon and attempt to drag to an invalid part of screen without enough money (These are combined as they can be checked together

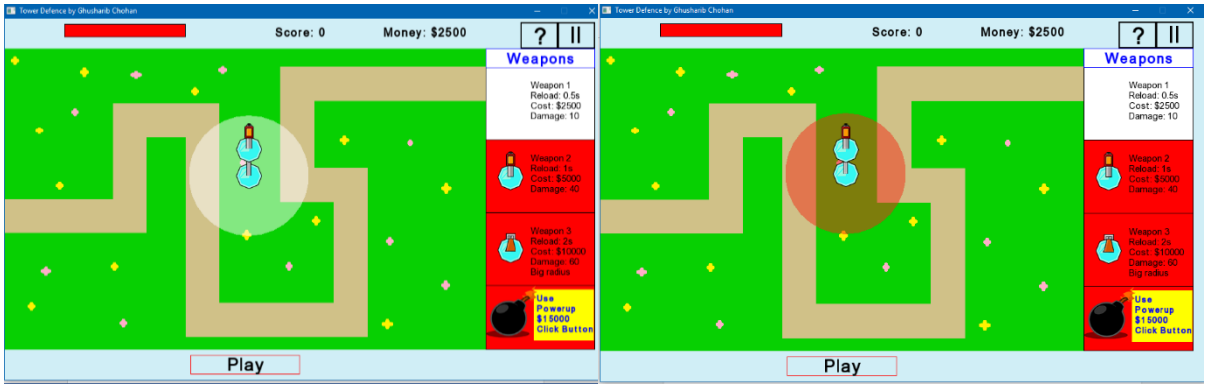


As can be seen in the screen above, the Weapon 2 has a red fill to it. This is the feedback given to the user showing they cannot buy that weapon (as a replacement to adding unecessary text messages). In addition, you are not able to drag the weapon anywhere so the weapon cannot be dragged to an invalid part of the screen.

[TEST PASSED]

Test 4) Click on weapon and drag on top of another weapon

The red radius shows that this weapon cannot be added to this position, giving the user feedback without a message but instead in the form of visual feedback. This means that weapons cannot be placed on top of each other.

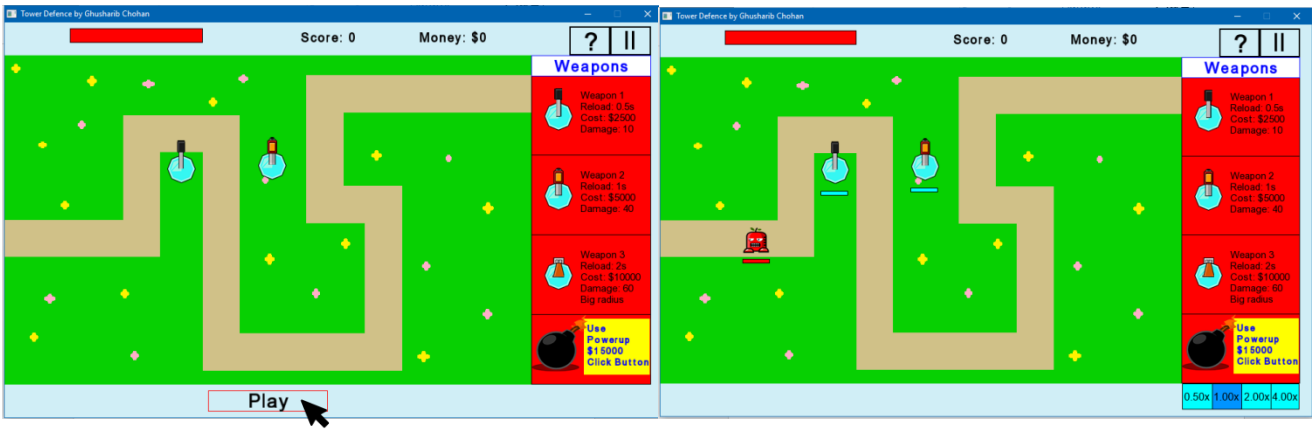


As can be seen in these two screenshots, the weapon can not be placed on top of another until it's base is just outside of the other weapon's base. This enables it to be consistent thoroughout.

[TEST PASSED]

Play Button

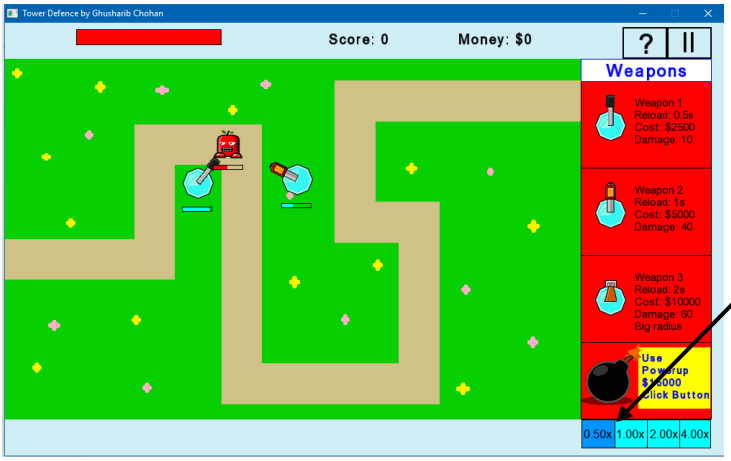
Test 1) Press the "Play" Button to run the simulation



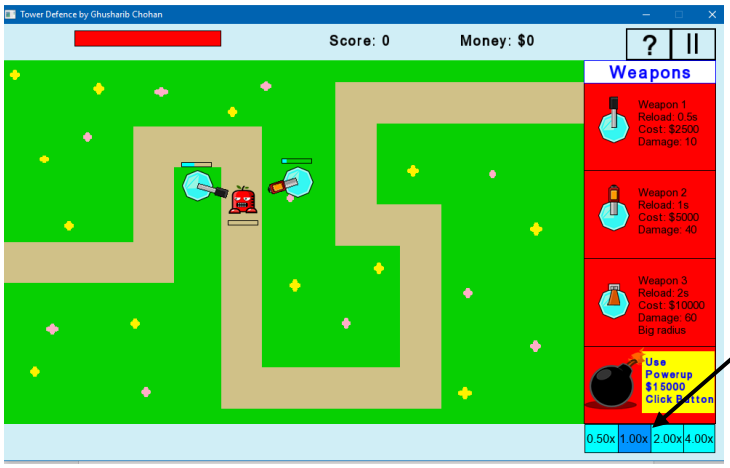
I press the Play button and the play button disappears. Enemies then are added to the map and begin to move. **[TEST PASSED]**

Speed Buttons

Test 1) Press the 0.5x Speed Button



The 0.5x Speed Button is now selected and the speed of the game is slowed down



Test 2) Press the 1.0x Speed Button

The 1.0x Speed Button is now selected and the speed of the game is sped up again returning to its original

Test 3) Press the 2.0x Speed Button

The 2.0x Speed Button is now selected and the speed of the game is sped up

[TEST PASSED]

I had now lost so had to restart the game

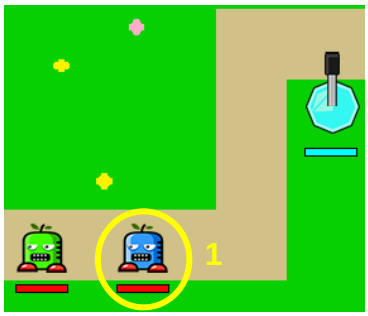
Test 4) Press the 5.0x (This was changed to 4.0x in the actual game) Speed Button

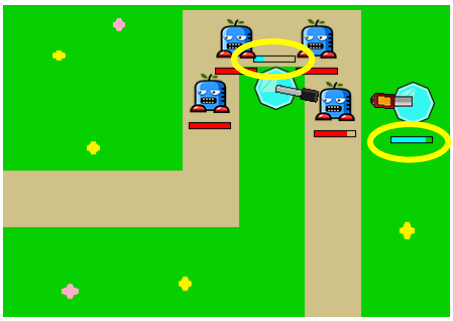
The 4.0x Speed Button is now selected and the speed of the game is sped up

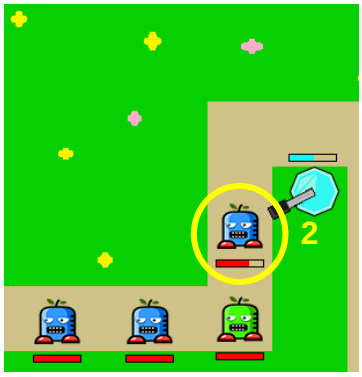
All the speed options work **[TEST PASSED]**

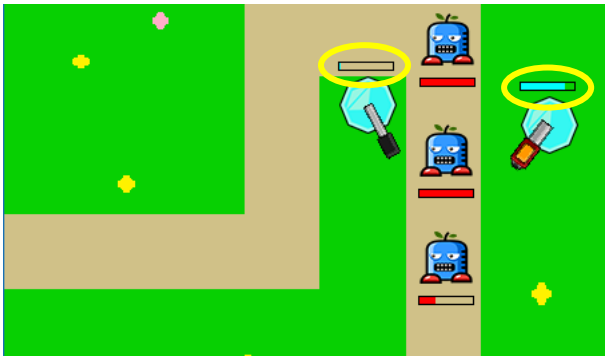
Health Bars

Test 1) Have an enemy come under fire from a weapon







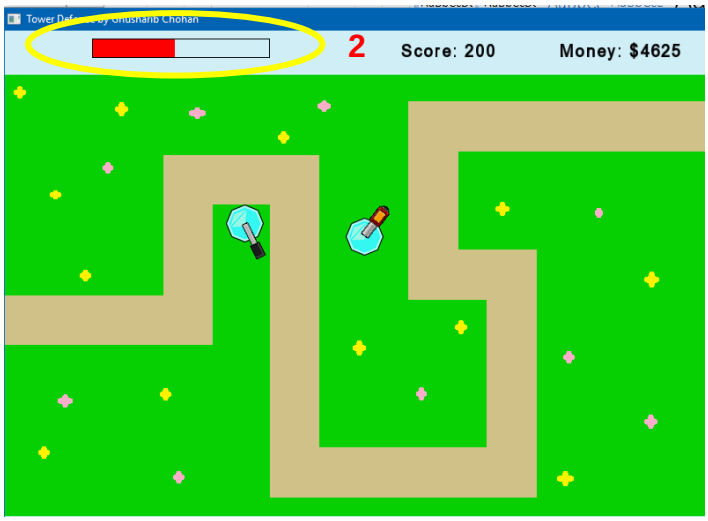


The health of the first enemy is full, but after being fired at by the weapon, its health goes down. This shows that the health bars are fully working

The reload bars are working as after they have fired at the enemy, they begin to reload and do not fire until they have fully reloaded

Test 2) Have Enemies reach the end of the map so that they damage the tower health, so that the towerhealth bar also decreases.

The enemy is reaching the end of the path with appoximately half of it's health left. This means it will be able to attack the tower.

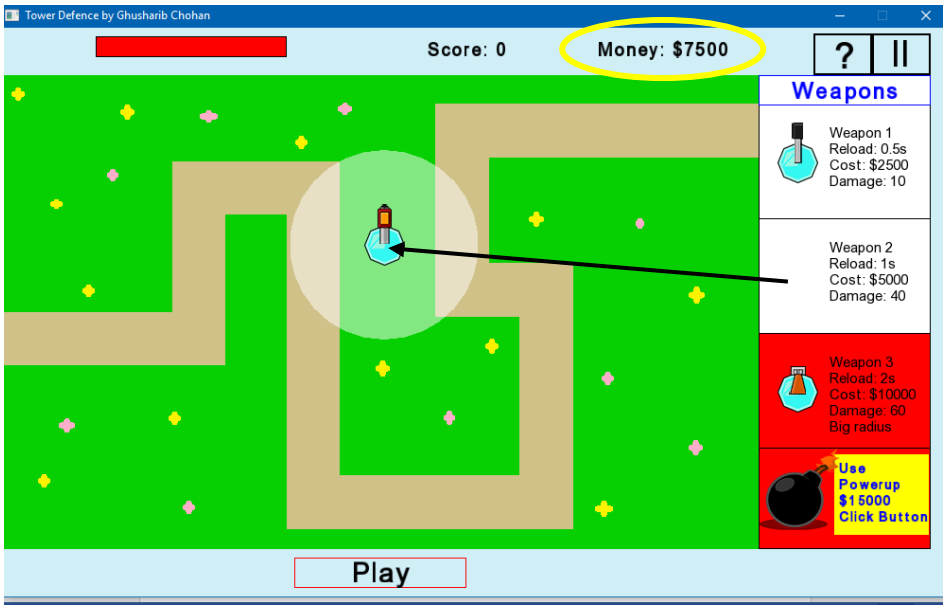


Upon reaching the end of the path, it is removed from the screen, and the tower health is decreased based on the health the enemy had upon attacking. The health abr can be seen to have decreased.

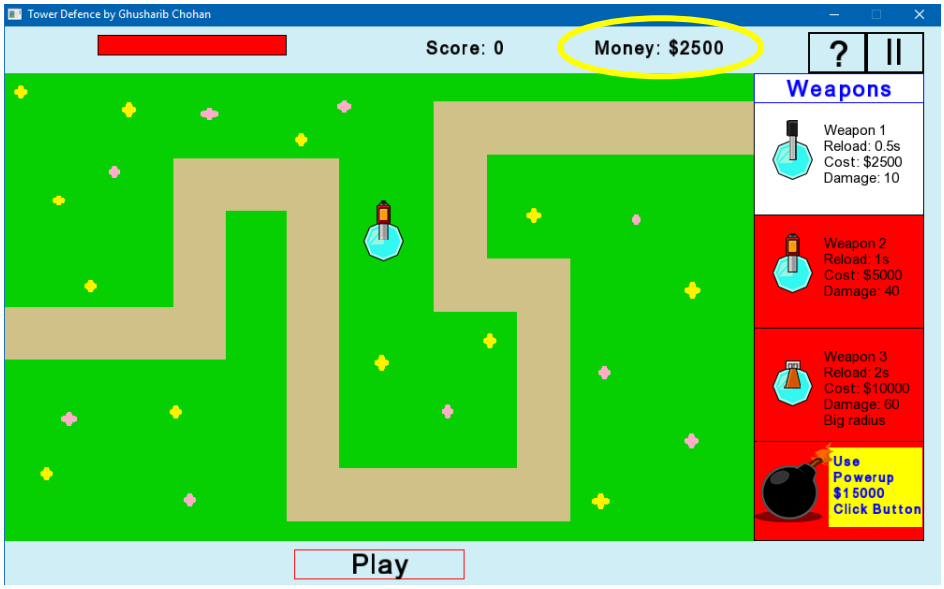
So All health bars and reload bars are working **[TEST PASSED]**

In-Game Money System

Test 1) Buy a weapon with enough money

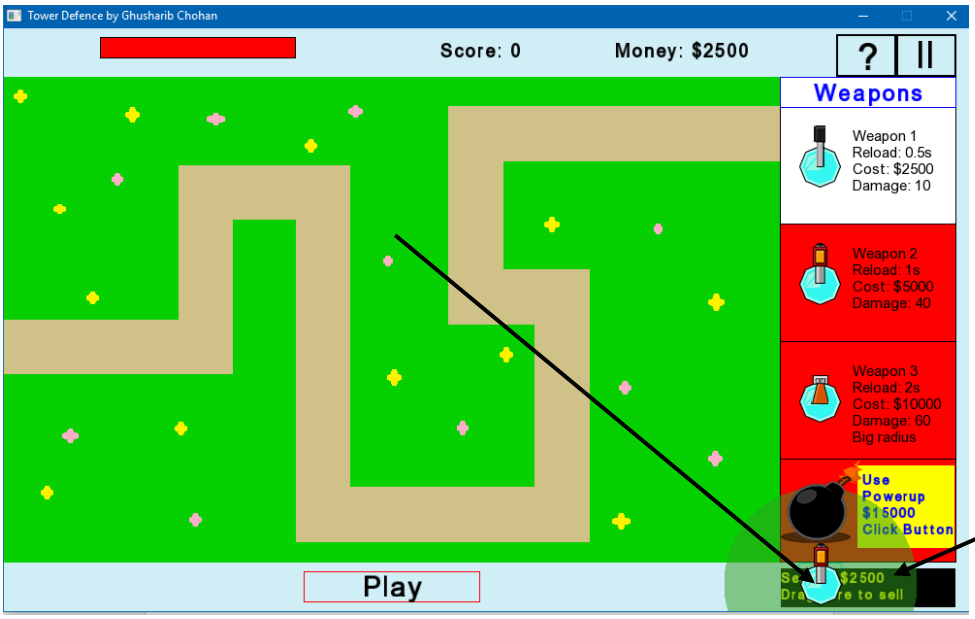


Weapon 2 is dragged onto the map when the user has enough money to buy it into a valid position (signified by the white radius)

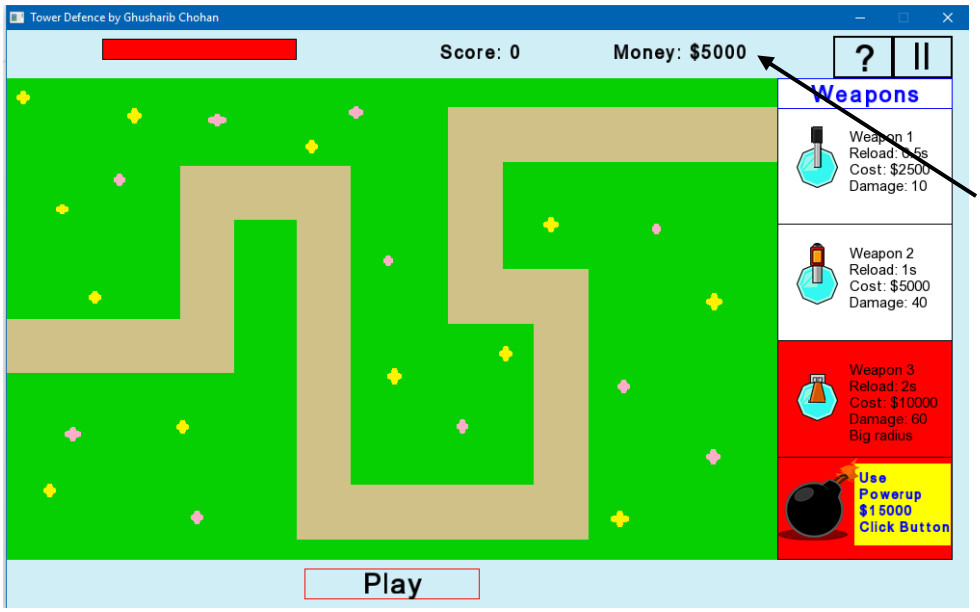


As can be seen, the money has decreased by $5000 – the cost of the weapon – and Weapon 2's background has turned from white to red signifying they can no longer buy a Weapon 2 (Which satisfies Test 2)

Test 3) Click and drag a weapon and sell it

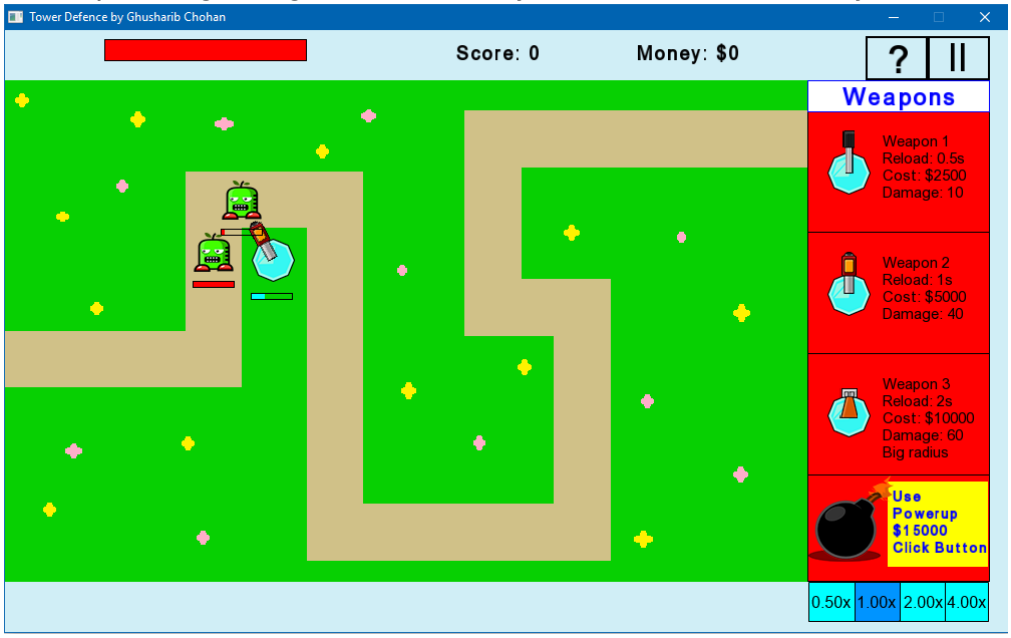


I clicked and dragged the weapon over to the black box where I can sell it. The weapon radius goes green signifying it is about to be sold if the mouse is released and says you will get $2500 back



The money then has $2500 added to it going back to $5000 showing

Test 4) During the game, an enemy should die and money should be added as a result



The health of the enemy being fired at is almost zero, and the money is currently $0

After the enemy dies, the user gets $875 for killing Enemy Type 2.

The money system works **[TEST PASSED]**

Score System

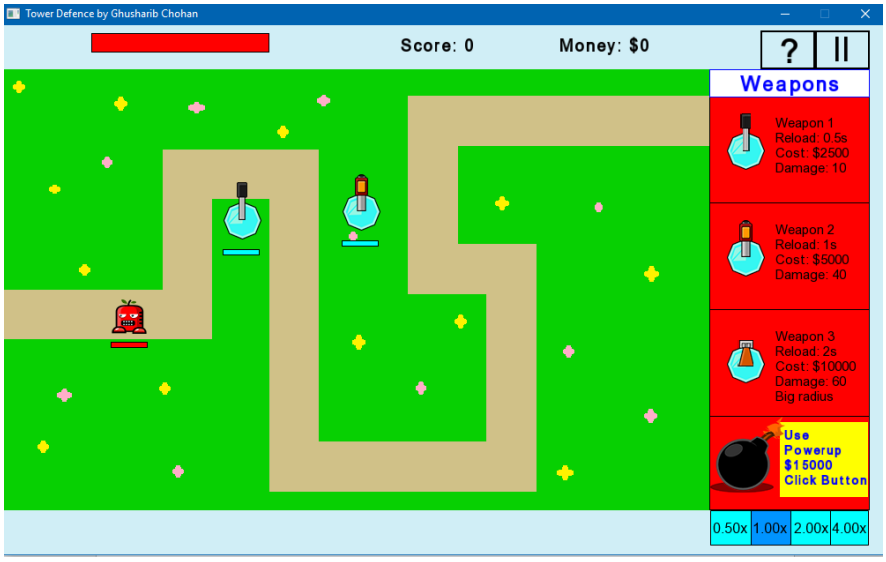
Test 1) Let an enemy die and score should increase. We can use the previous screenshot to see that it does work. The score has gone from 0 to 50.

[TESTS PASSED]

Emergency Powerups

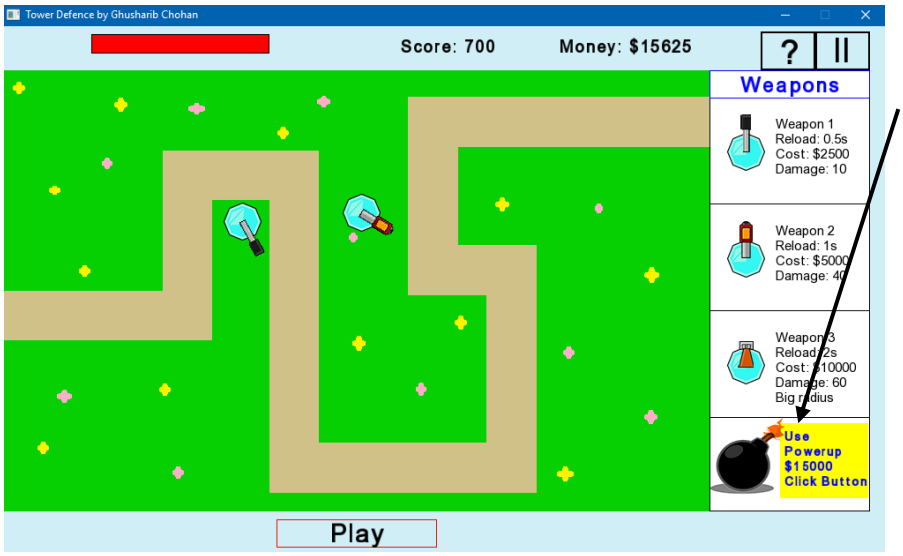
I will combine the 4 tests for this as one as they can be tested as I play the game. The tests are as follows:

1) Use Powerup with enough money
2) Use Powerup with enough money but already used
3) Use Powerup without enough money
4) Use Powerup without enough money and already used



Initially, we can see that the background for the powerup is red. This, as suggested before, shows the user they cannot afford it. Attempting to use the powerup leads to no reaction as they cannot afford it and so cannot use it (note this is a button, not a click and drag)

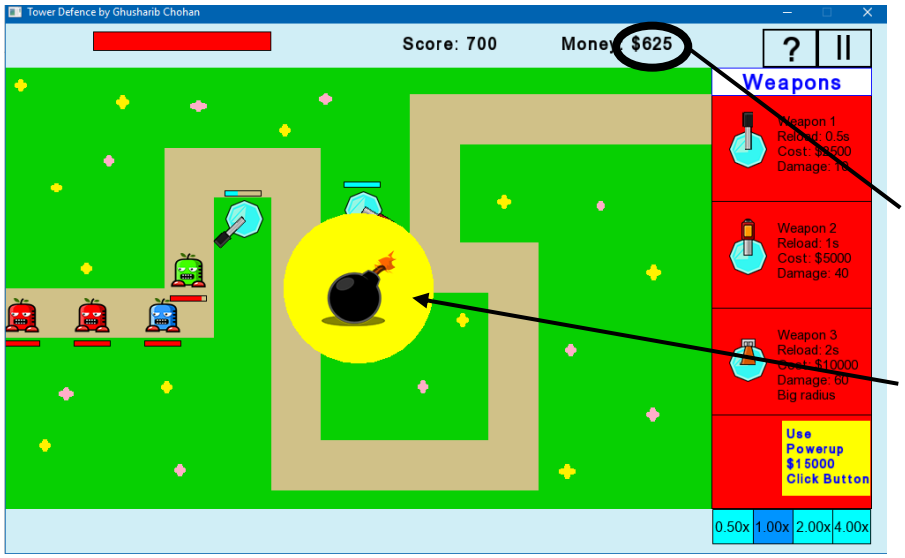
I continue to progress through the game and it takes time before I can finally afford the powerup. This is as it should be to ensure that the user has played for a significant amount of time before being able to use the powerup.



When I can afford the powerup this happens:

The background has gone white showing I can now afford it and can use it. However, the current wave has ended so I do have to start another wave to use it – you can only use the powerup when the play button has been pressed

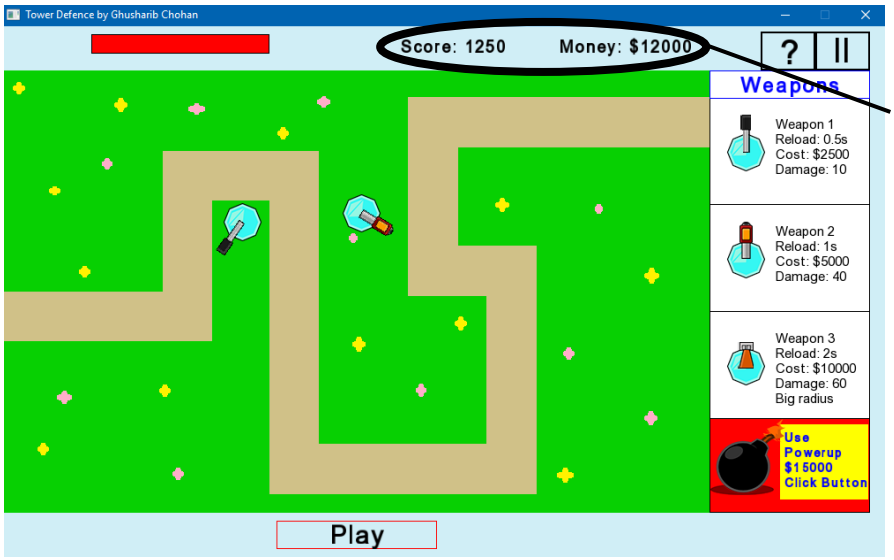
I then proceed to click on the Use Powerup Button and as expected, the animation .begins:



There are a few things to note here for the tests.

1. The money has decreased as a result of using the powerup
2. The powerup animation is running

The animation runs all the way through and then this is the result



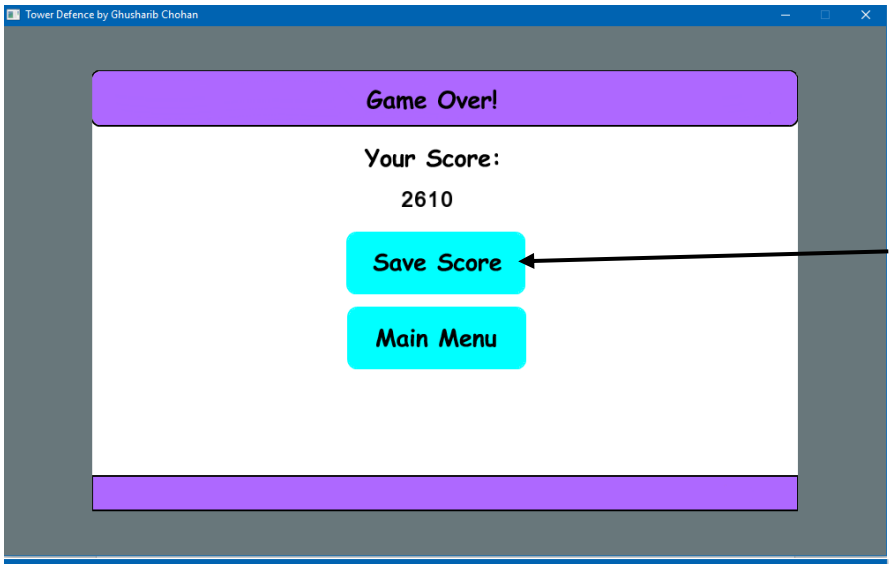
The score and money has now been added to the user's running totals from killing all the enemies. All the enemies from the screen have been removed and the game returns to the add weapons loop.

Now that the weapon has been used, pressing on the button does nothing regardless of whether or not you can afford the powerup. We can therefore confirm that the powerup works as it should **[TESTS PASSED]**

High Scores

Test 1) Lose the game

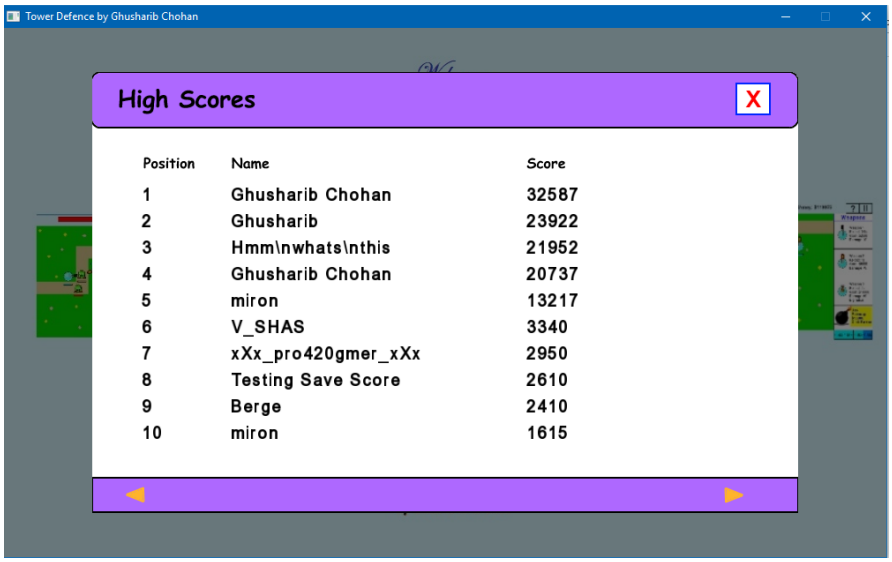
I purposely lose the game I was previously playing and this screen appears. Here I have two options – the first is the one that is being tested



I click on the “Save Score” Button to save my score

I am greeted by a screen requesting me to input my name

I enter the name “Testing Save Score” to show where in the file the score ends up. I then click on the “Save” Button



The score ends up in the correct position in the high scores table.

It should also be noted that attempting to break the file with “\n” (delimiters for when a new line occurs) has no impact on the durability of the high scores file and you are unable to add commas, as can be seen by the Name in Position 3.

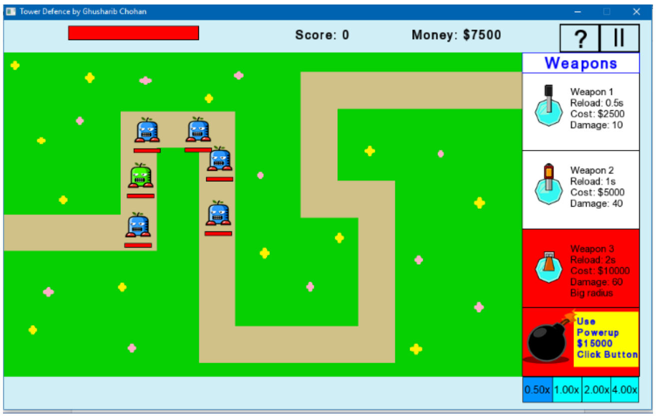
The highscores option came up automatically when I lost the game as was required and therefore this works perfectly

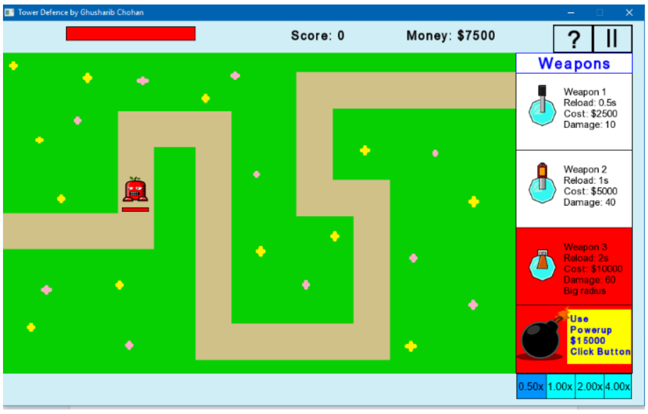
[TEST PASSED]

Random Games

Test 1) Play the game three times and see if you get a random game each time

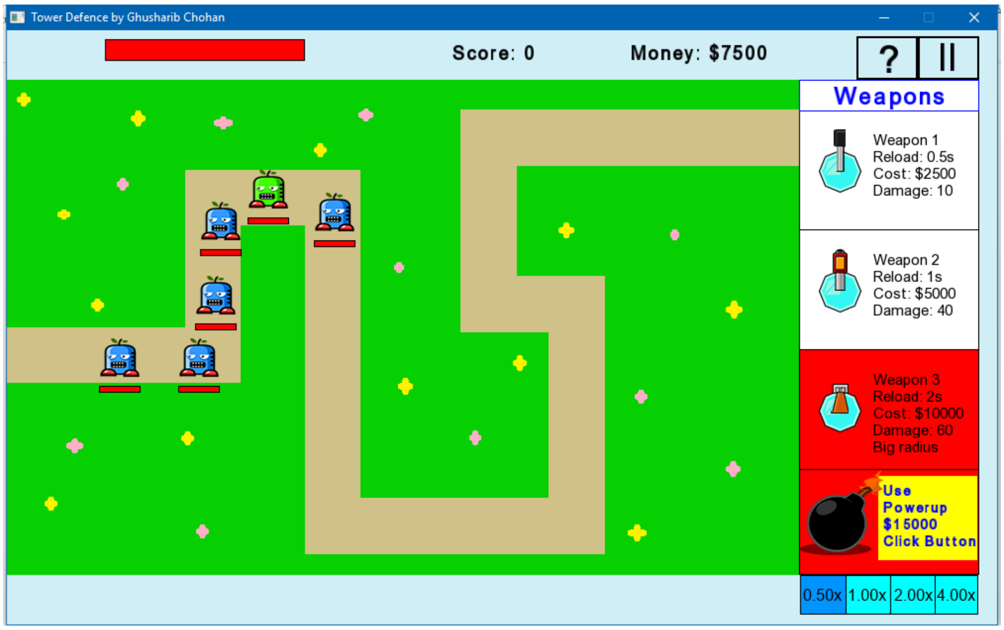
Below are screenshots of three games and the enemies I got in the first wave of each game:





As can be seen, I get three completely different games each time.

On my fourth game, I got the same enemies as Game 1 but in a different order, suggesting that enemies are added truly randomly: **[TEST PASSED]**



Game Endless Until Failure

Test 1) Play Game 3 times

There is no way of showing that the game is endless until failure through screenshots, but each time, the game would not end until the tower health reached zero, so I can safely say the test has passed

[TEST PASSED]

Information on Next Wave of Attackers

[TEST FAILED]*

***This test failed as this was later removed in the second problem specification that was accepted by my stakeholders, after my stakeholder Nathan decided that there was no real point in having a next wave information bar as it offered nothing to the game.

This concludes the tests that I had to carry out

Final Testing Grid (Overview of previous pages):

Test Item	Passed/Not Passed
Click on Help Button (See Help Button Testing Section Below)	**Passed**
Click on High Scores Button	**Passed**
Click on New Game	**Passed**
Click on a weapon of choice and drag it to a valid part of the screen (i.e. to the game grid) with enough money	**Passed**
Click on a weapon of choice and drag it to a valid part of the screen (i.e. to the game grid) without enough money	**Passed**
Click on a weapon of choice and drag it to an invalid part of the screen (i.e. just outside of the game grid)	**Passed**
Click on a weapon of choice and drag it on top of a previous weapon	**Passed**
Press the "Play" button to run the simulation	**Passed**
Press the 0.5x speed button	**Passed**
Press the 1.0x speed button	**Passed**
Press the 1.0x speed button	**Passed**
Press the 2.0x speed button	**Passed**
Press the 1.0x speed button	**Passed**
Press the 4.0x speed button	**Passed**
During the gameplay, one of the enemies should come under fire by a weapon	**Passed**
During the gameplay, some enemies should reach the tower and attack the tower	**Passed**
Buy a weapon (using click and drag feature) with enough money	**Passed**
Buy a weapon (using click and drag feature) without enough money	**Passed**
Click on a weapon already placed in the game and click on "Sell Weapon"	**Passed**

During the gameplay, one of the enemies should die	**Passed**
During the gameplay, one of the enemies should die	**Passed**
During the gameplay, buy a powerup from the weapons menu using the click and drag feature (with enough money and not bought before)	**Passed**
During the gameplay, buy a powerup from the weapons menu using the click and drag feature (with enough money but bought before)	**Passed**
During the gameplay, buy a powerup from the weapons menu using the click and drag feature (without enough money and not bought before)	**Passed**
During the gameplay, buy a powerup from the weapons menu using the click and drag feature (without enough money and bought before)	**Passed**
High Scores	**Passed**
Endless Game	**Passed**
Random Game	**Passed**
Click on Help Button	**Passed**
Click on close button on help screen	**Passed**
Information On Next Enemies	**Failed**

Problem Specification Check List (Met and Not Met)

Problem Specification Point	Met/Not Met	Evidence
Game should be endless until failure	Met	Page 143
Speed up and Slow Down	Met	Page 132
Increasing Difficulty	Met	Page 142
Emergency Powerups	Met	Page 138
Health Bars, Weapon Info	Met	Pages 129 – 131 and Pages 133-134
Upgrades	Not Met	N/A
In game Currency and Score	Met	Pages 135 – 136
High Scores	Met	Pages 139 – 141
Start Button	Met	Page 131
Menu and Instructions	Met	Pages 127 - 128

User Feedback

I decided to create a Google Forms form for the user's to fill out for the purposes of the final testing. For this, I emailed all of them a copy of the game and the link to the form. The copy was sent in a .msi format so that it could be directly installed onto the target computer with limited input.

The individual feedback forms can be found as part of Appendix E.

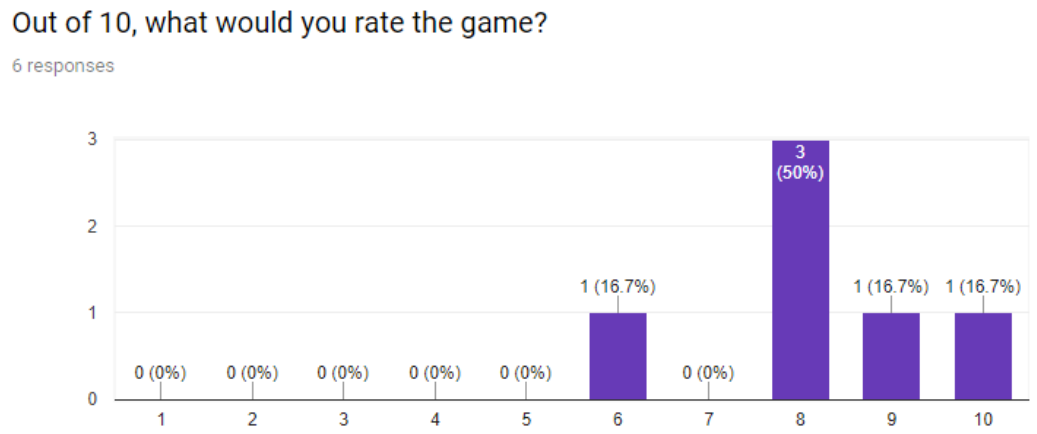
The questions asked were as follows:

1. **What is your full name?** – I required this so that I could ask them follow up questions if need be and so needed to know which of my stakeholders was giving the responses
2. **Out of 10, what would you rate the game?** A rating out of 10 is quantifiable and allows me to easily work out whether or not the user enjoyed the game. This was therefore the easiest and best way of understanding whether or not the user enjoyed the game.
3. **What is the reason for your above rating?** Once I have a rating, I would like to know why they think this is x out of 10. This is because this will allow me to understand the strengths of my game as well as any weaknesses very quickly.
4. **Did you notice any bugs?** If I later on make any updates to the game, I will need to remove bugs that they may have noticed. These bugs are not always noticed by me, the developer, as I am not playing the game the same way my users are and therefore, it is important for me to ask this question.
5. **How long did you spend playing the game?** This is another quantifiable question allowing me to further understand how much they enjoyed the game. I previously stated that I expect users to play the game for 30 minutes for my game to be considered successful so this will be key to my understanding.
6. **Accepting the problem specification.** This has checkboxes for the entire problem specification. Users can tick off all the boxes they feel have been successfully achieved. This will help me see whether or not the users feel as though the things they requested in the game have been met or not.
7. **Are there any additions you would like to make to the game?** Once again, if I were to update the game at a later stage, I would like any suggestions to be taken into consideration, and my users are the best people to ask.
8. **Any final comments?** This is for their own general use and is not a requirement to be filled out.

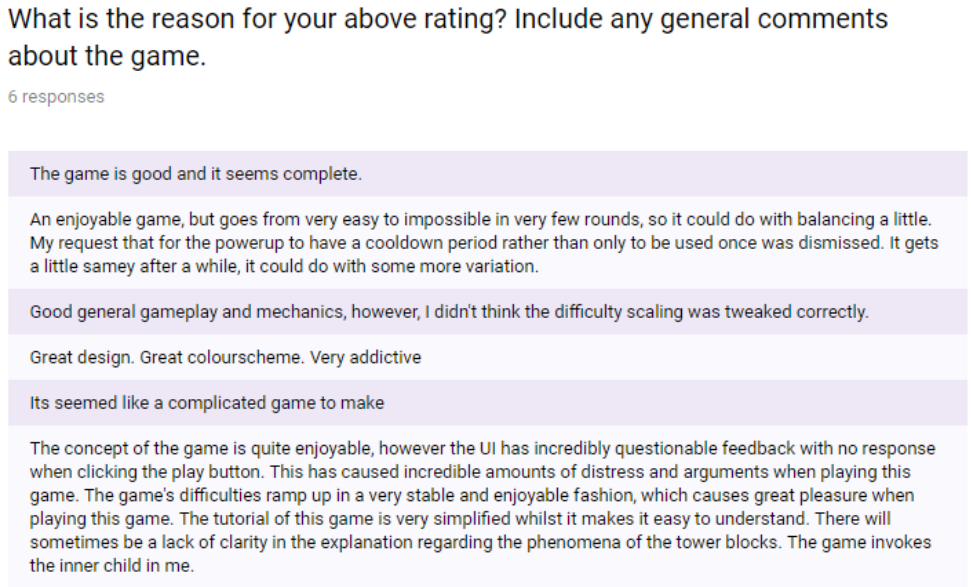
Summary of Responses

As stated before **Appendix E** contains all the individual responses.

Below is a summary of the responses and what I get out of them:



My modal rating for the game was 8/10 with 50% of my stakeholders giving it 8 out of 10. There were some unusually high and unusually low ratings as well including a 10/10. This meant I had an **average rating of 8.2/10** which can be considered to be a very good rating by which I am pleased.



In terms of reasons for these ratings, everyone seemed to enjoy the game (with one even suggesting it brought out their inner child). However, some did suggest that the balancing of the game was not quite there as per their requests (despite the improvement of Weapon 3 back in Iteration 6 of the devlopment) and that there were a few issues with the User Interface. However, these seemed to be specific to one

user only and there was no general consensus on what was good and bad. Everyone had different opinions on the strengths and weaknesses of the game.

Whilst playing the game, did you notice any bugs that you think need fixing?

6 responses

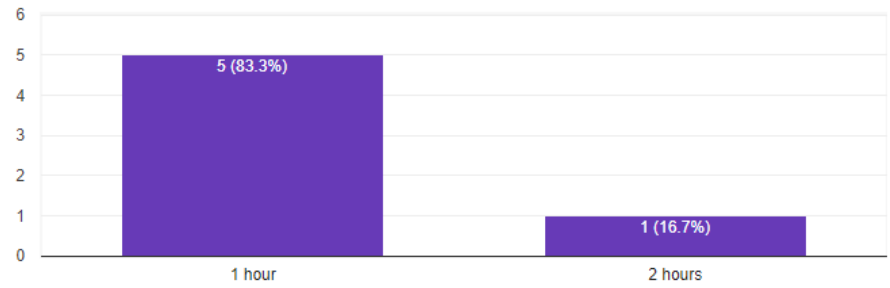
The dragging stops when you move the mouse out of the window
Validation required on highscore name input - any length is allowed causing overflowing layouts. Changing the game speed does not change the bullet speed, so an advantage is gained by playing in slow motion.
There was a micro-stuttering that occurred during 0.5 times speed gameplay. Also, the current speed of gameplay resets after every level.
No
No
The start button doesn't respond quickly.

One noted that dragging stopped when the mouse moved out of the window. However, this was not a bug, but rather a feature of the game. If the mouse exits the game screen, the weapon is in an invalid place and therefore cannot be placed outside of the screen. This means that the game is working as it should and so this is not a “bug” that needs consideration.

Another suggested overflow in the highscore name input. This was a valid bug which will be fixed in future versions of the game. However, micro-stuttering was only pointed out by one user and it may be down to the computer they were running the game on. The same can be said about the start button.

How long did you spend playing the game?

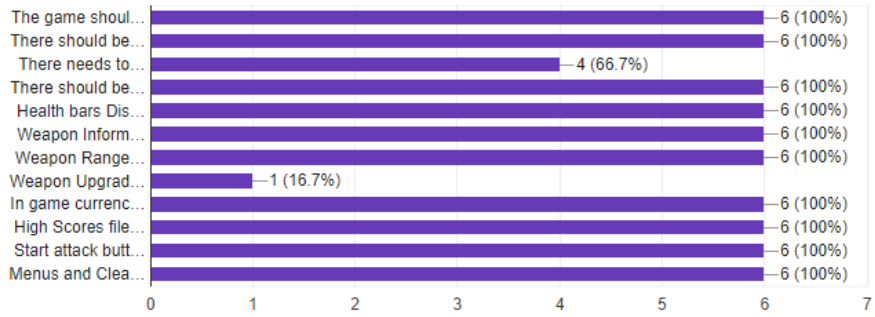
6 responses



All of my users played for longer than I expected them to. 5 out of 6 (83.3%) played for one hour and the other user ended up playing for two hours! This was very unexpected and pleasing and shows that my stakeholders were not only satisfied with the game, but also very addicted to it!

Below is the problem specification which you may remember from the early stages of the development which you signed off. This is list of requirements which you felt needed to be met for the game to be considered "successful". Please tick all the options below which you feel have been met.

6 responses



Most of the problem specification appears to have been achieved. However, 2 out of 6 of my stakeholders felt that the game was not balanced and most of my stakeholders did not feel as though weapon upgrades were a thing in my game. I agree with this as I forgot to add upgradability, but this was partly because upgradability was something I felt would ruin the balance of the game even further, and therefore is not a feature that I will be willing to add to the game soon. Upon asking for further feedback, many of my users agreed that weapon upgradability was not a feature they felt was required very urgently. This concludes the main parts of the survey.

I have now received user feedback and carried out all the testing parts as were set out in the design stage.

Evaluation
Ghusharib Chohan 605

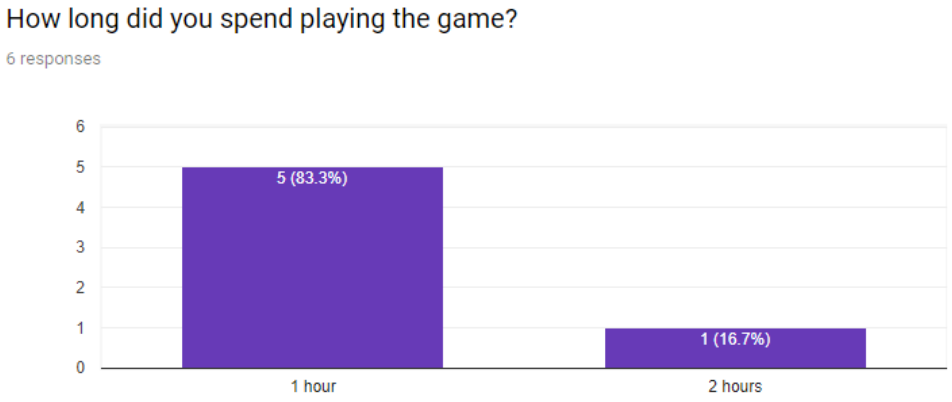
Testing the robustness of the game

Back in the development stage, I had during development testing which followed my test plan from the design stage. I have then carried out post-development testing and user testing as can be seen in the pages before.

Measuring my success using the success criteria:
Back in the Analysis Stage, I made a success criteria which I thought would help me understand whether or not the game has been successful. The criteria was as follows:

4) The game needs to be entertaining. Users should be able to play for the times they have stated in their interviews (an average of half an hour) and not get bored in that duration.
5) The game should run fairly smoothly.
6) Users should feel that their issues in terms of the problems usually faced in tower defence problems have been addressed. The biggest problem needs to be that users should not be able to power through all the levels and equally not be facing excessive difficulty. They will need to be quizzed on the "balance" of this tower defence game.

The first criteria is that the game needs to be entertaining. Users told me that they would be willing to play around half an hour if the game was considered to be entertaining and so I used this as a benchmark for my game. This was then asked as a question as part of my user feedback form and this was what was said:



5 out of 6 of my users played for 1 hour and one of my users even played for 2 hours! This has breached my expectations of half an hour as 100% of my users played for longer than half an hour. This suggests that the game was addictive and that they genuinely found it entertaining albeit there were a few issues they had with

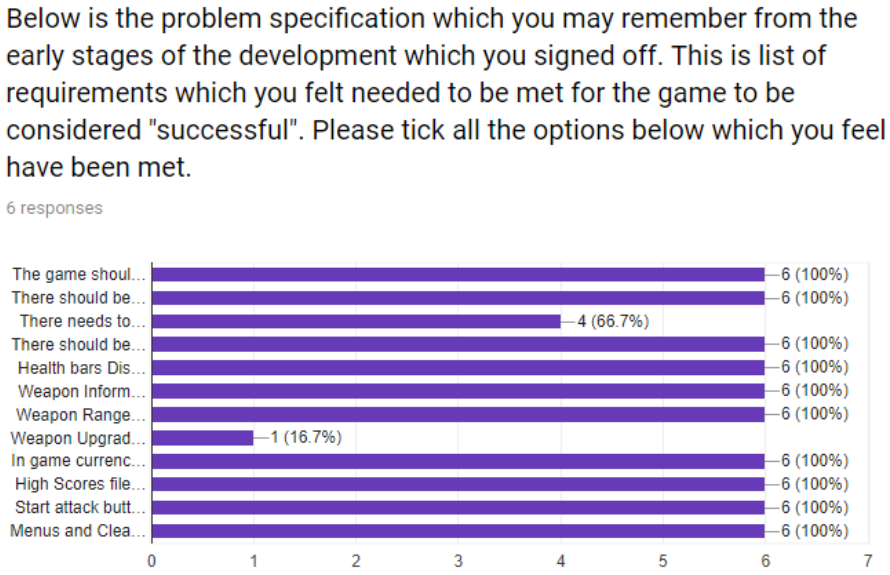
the game. As a result, it is fairly safe to say that this can be ticked off the success criteria as being well met (100%).

The next is in regards to the smooth running of the game. The game will progressively get faster and faster as it runs through the game, as part of the problem specification. This means that processes occur many, many times every frame and this can cause significant lag. Therefore, my game needs to be efficient enough to cope with such requirements.

In the first user feedback back in iteration 6, the game was thought to be very laggy. However, upon creating a release executable – the file that was played at the end of the development – the game seemed to be rid of such problems. The game no longer lagged and tended to run at a healthy 60 Frames per second even on the worst of computers. Note, that I had capped the frames to run at no more than 60FPS.

However, one of my users mentioned micro-stuttering when running the game at 0.5x speed which is an issue, and therefore I cannot say that this was met fully. In addition, most of my users did not reach a stage where lag would have become a significant factor and therefore, this was not a brilliant test of the robustness. Therefore, I would say this criteria was met with 85% success.

The final criteria was balance in the game. This was the most important criteria as the entire problem with previous Tower Defence Games is that they are either too easy or too difficult. My game addresses this issue by creating a random game each time, being endless until failure and using exponential increase in money and difficulty. Once again, I asked my users the question as to whether they felt the game was balanced or not and the responses came as follows:



As can be seen, 4 out of 6 (66.7%) felt that the game was balanced, but 2 did not feel it to be so. The reason for this was that they felt that the game probably got too difficult too quickly. Even though the money was enough to just get you through the levels, as it should be, they probably wanted the game to last longer so that they can deploy more tactics. Therefore, this criteria can only be passed by 66.7% and isn't fully met.

This means that according to my success criteria, the success of my solution is around 84% which is a very good percentage. However, there are clearly areas which need improvement for my game to be considered fully successful.

Usability Features

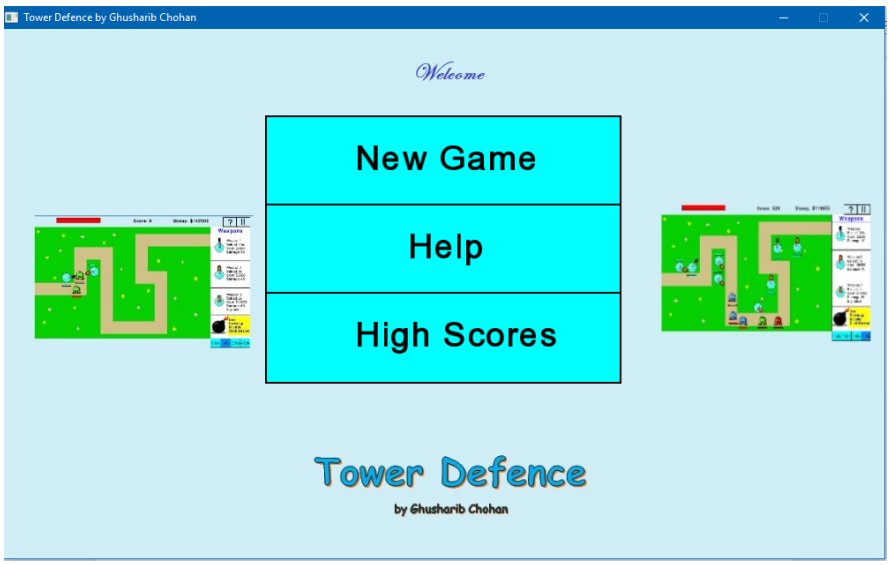
For my game, I decided on some usability features that needed to be implemented back in the design stage. These were as follows:

- Help screen available at all times
- Error textboxes whenever the user makes a mistake
- Speed buttons
- Clear design with big buttons

Feature 1 – Help Screen

The help screen was a feature that was 100% required. My users often did not know how to play the game and by having a help screen available, they were easily able to understand how the game works and how they should play it. This also needed to be intuitively placed. Therefore, this was how it was displayed:

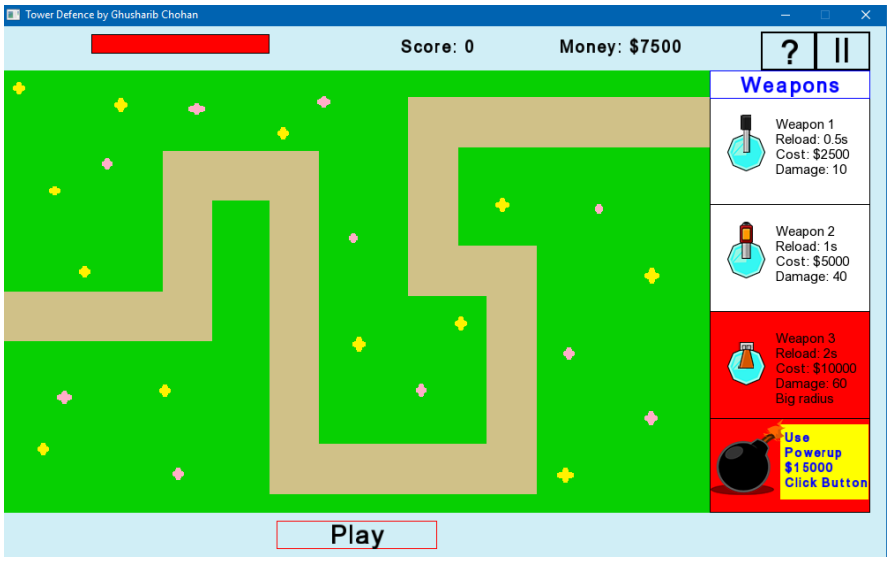
1) When the user starts the game, they are met with the main menu. This is the first place where the Help Screen become available.



The "Help" Button appears with all the other buttons (New Game and High Scores) so that the user is not left searching for a help button. It is intuitive that they need to click on it if they face any difficulties in the game.

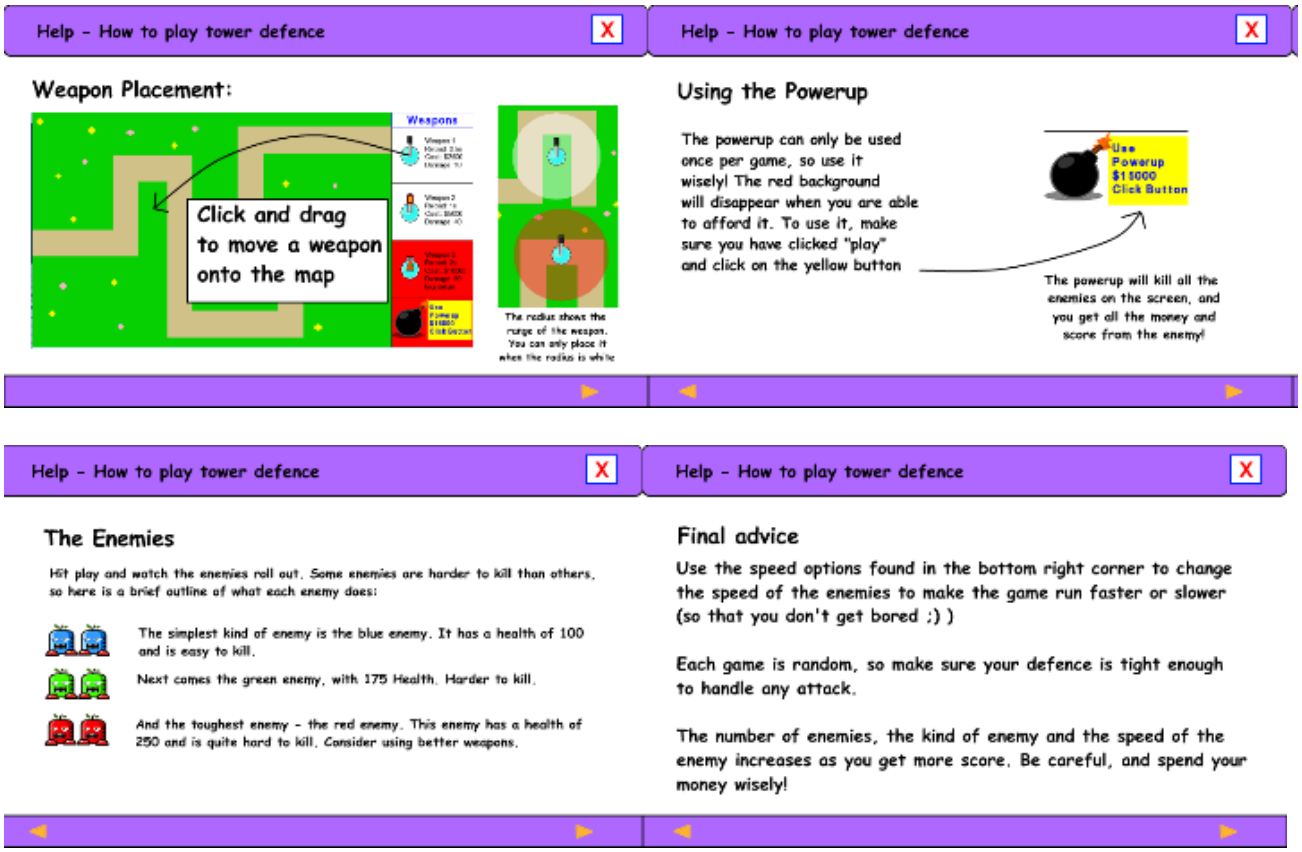
However, the user won't always be on the main menu and so the help button needed to be available during the game as well.

2) When the game is being played, there is a Question Mark in the top right corner. This is as in all applications the standard for short hand help buttons. The user will know to press this button if they require help during the game.



The intuitive button position on the game screen makes it easy to use and access. It is right next to the pause button.

The help screen itself has the screen as follows



The screens contain valid information. The first screen shows the user how to add weapons and once added, they need to click on the play button to begin playing the game. The second shows what the power up does and informs the user of its one

time use feature. The third gives information on the enemies and the final screen tells the user about speed buttons and how the game will advance. All of this information is useful for the user to know. Another notable feature of the help screen is the navigation buttons on the bottom:

These are to help the user navigate from one page of the help screen to the next. The previous button also ensures that the user can navigate back to a page that they may not have finished reading.

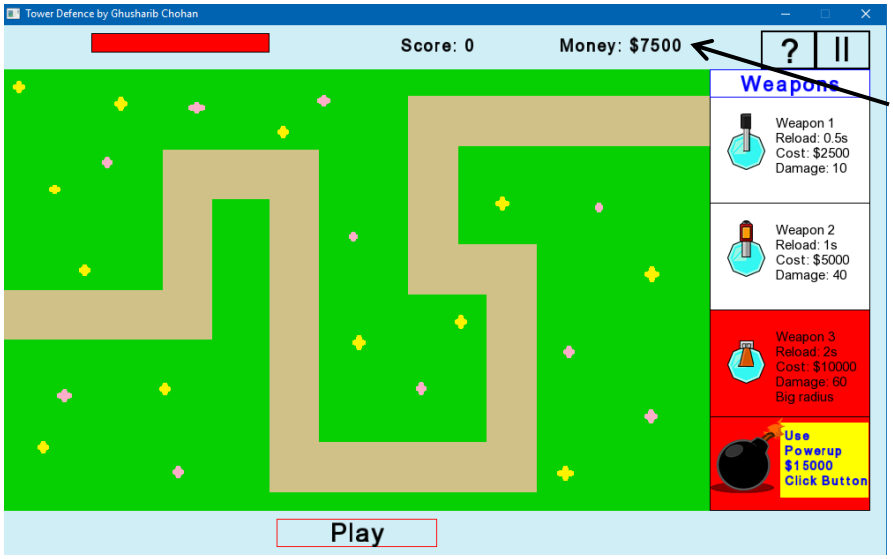
Another key feature is the pause button itself. Upon pressing pause, the gameplay stops and does not move. This ensures that the user can pause the game if something important comes up in the real world and not have to worry about losing the game. This pause feature also occurs upon pressing the help button, so that while they are looking at the help screen, they do not end up losing the game. This is very beneficial for the user.

Feature 2 – Error Messages

During the development stage, I actually decided not to implement error messages but instead other more intuitive ways of showing the user they are doing something wrong. This was particularly useful as it ensured that I did not waste valuable memory on displaying text values. These intuitive methods can be seen below:

1) Buying weapons.

Each weapon has its own cost associated with it. Weapon 1 is $2500 while Weapon 2 is $5000. The user also has money e.g. at the beginning, they start with $7500. Initially, if the user could not afford a weapon, they would have received an error message telling them that they did not have enough money to buy it when they would attempt to add it to a map. However, this would mean they would have to drag it all the way onto the map before getting any feedback. A more intuitive way was having backgrounds to the Weapons which told the user whether they could afford it or not. A white background meant that they could afford it while a red background meant that they could not afford it. In addition, they are not allowed to move a weapon onto the map unless they can afford it, making the game simpler to understand:



In this first screenshot, the user has $7500 and hence can afford either Weapon 1 or Weapon 2. Therefore, the two weapons have white backgrounds. However, Weapon 3 and the power up have red backgrounds.

Upon purchasing a weapon 2, the user no longer has enough money to purchase a Weapon 2 and therefore Weapon 2's background also turns red.

This also gave real time feedback as when the play button was pressed and the simulation ran, as soon as they could afford Weapon 2, the background would become white again. This was an added bonus of this usability feature.

2) Weapon Dragging

If the user was to attempt to add a weapon to a place outside of the map itself or on top of the path, the user would require feedback telling them that they cannot place the weapon there. Error messages are fine, but on places near the boundaries of where the user can and cannot place a weapon, this can become frustrating. Therefore, it is easier to change the colour of the radius to show where a user is hovering over an area where they can place a weapon and where they cannot as can be seen in the following screenshots:

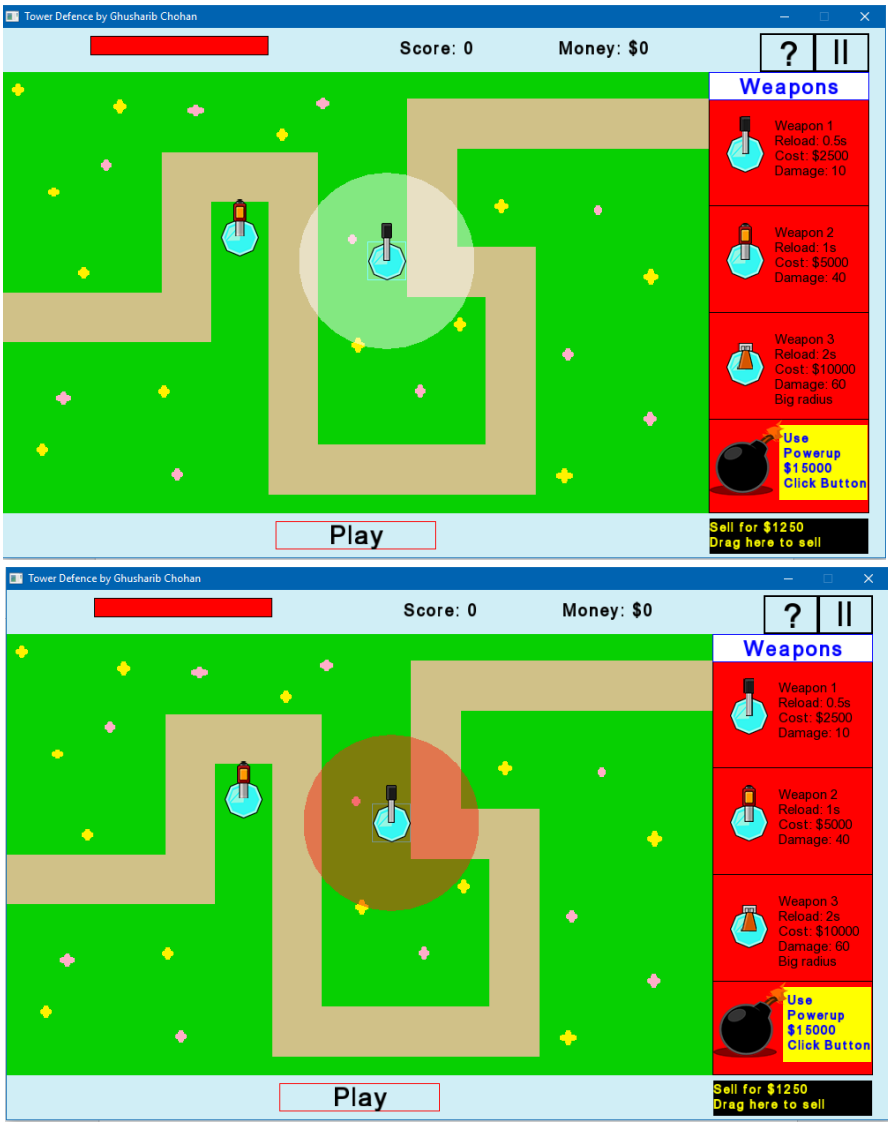
Attempting to place the weapon on the path causes a red radius meaning that the user cannot place it there. This is intuitive.

A white background shows the user that they can place the weapon there and is much simpler than using textboxes.

In addition, the radius colour changes when hovering over the sell button:

The green colour intuitively means money and shows the user they are about to sell the weapon. This makes sure that the user knows what they are about to do before confirming the sale.

These features are intuitive and make the game much simpler. If textboxes were used, they would cause much frustration near boundaries such as those shown below:



The weapon has probably barely moved a pixel but a red radius has appeared. If this was done using textboxes, the user would have been very much frustrated as they would not have been able to get instant feedback as is made possible using the changing colour of the radius.

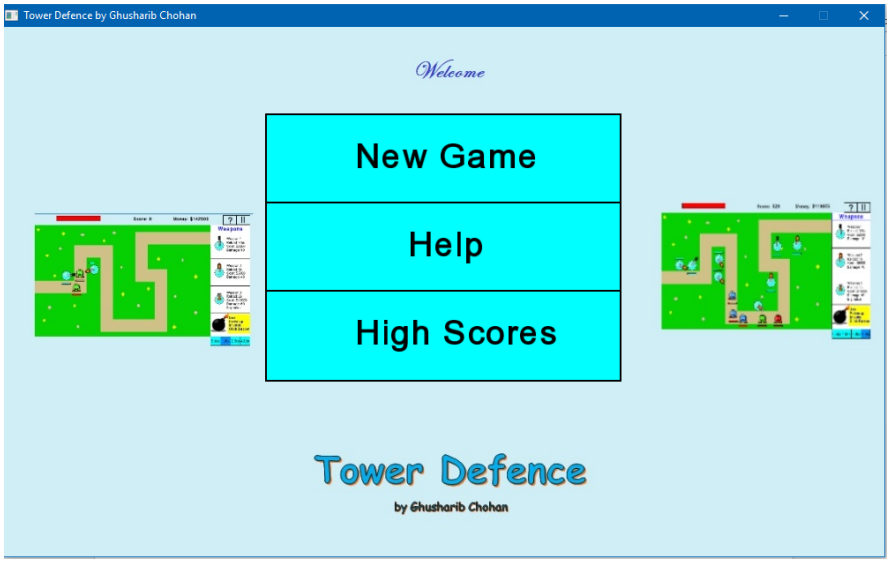
Feature 3 – Speed Buttons

These were placed in the bottom right corner which is an intuitive place for the speed buttons:

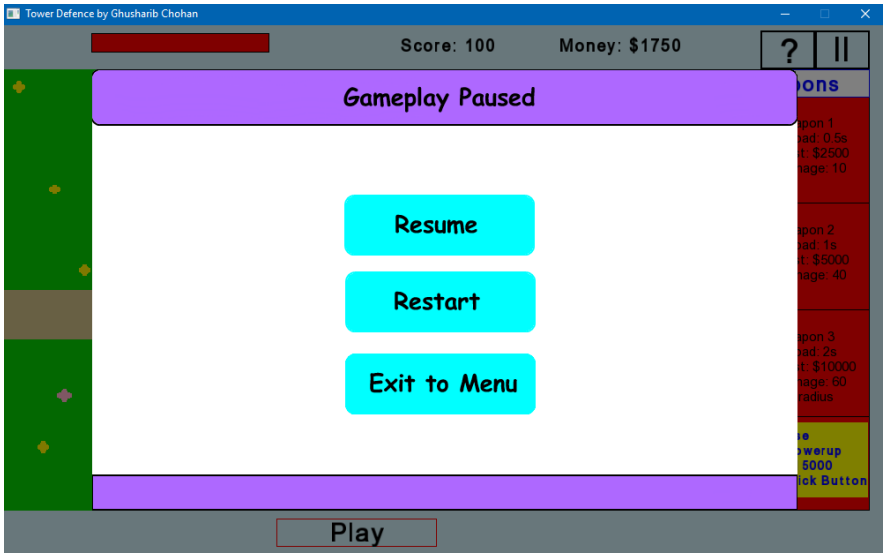
Referring back to the testing stage, these did work as required and were a usability feature that work.

Feature 4 – Big Buttons

If we look back at the Main Menu, Big Buttons are definitely there:



The same can be said for the pause screen and other screens of the game:



Therefore, all my usability features have been successfully implemented and make the game easy to use as required.

Maintenance of the game

A measure of how well I coded my game is how much change I need to make to the code when adding to it and fixing bugs. If the program has been well coded, there shouldn't be too much requirement to change major parts of the code.

The biggest benefit of my code is that it is a modular design. My code has been broken down into smaller parts and each of these parts has its own block of code associated with it. This means that I know which part does what and where I need to insert new code. This gives the benefit of simplicity for future maintenance as each of my module is self-contained and independent of other functions. This also means it will be easy to add any new modules of code into my game in the future.

```
  1   #include "GAME.H"
  2
  3   void Game::processPreGameEvents()
  4           {
225
226   void Game::addNewWeapon(sf::Vector2f position, sf::Vector2f turret, int WeaponType)
227           {
270
271
272   void Game::invalidPlace()
273           {
282   void Game::preGameRender()
283           {
```

The above picture shows the various modules of code which are called when processing events while the user is adding weapons.

In addition, I have also used classes which will also make it a lot easier to maintain the game. This is because each Weapon Type and Enemy Type has its own class, an object of which can be created and added to a vector of the general type class:

```
  1   #ifndef WEAPONS_H
  2   #define WEAPONS_H
  3   #include <SFML/Graphics.hpp>
  4   #include <string>
  5   #include <iostream>
  6   class GeneralWeaponClass
  7   {
250
251   class Weapon1Class
252   {
370
371   class Weapon2Class
372   {
490
491   class Weapon3Class
492   {
611
612   class PowerUpClass
613   {
717   #endif
718
```

The picture to the left shows the Weapons Classes for the game. As each weapon type has a separate class it allows me to change the abilities of the different weapon as can be seen on the next page.

```
class Weapon1Class
{
    public:
    Weapon1Class()
        : BaseSprite(), BaseTexture(), TurretSprite(), TurretTexture()
        {
            if(!BaseTexture.loadFromFile("Images/MainBase.png"))
            {
                //Handle Loading Error
            }
            BaseSprite.setTexture(BaseTexture);

            if(!TurretTexture.loadFromFile("Images/Weapon1Top.png"))
            {
                //Handle Loading Error
            }
            TurretSprite.setTexture(TurretTexture);

            if(!BulletTexture.loadFromFile("Images/Weapon1Bullet.png"))
            {
                //Handle Loading Error
            }
            BulletSprite.setTexture(BulletTexture);

            Border.setSize(sf::Vector2f(180, 120));
            Border.setOutlineColor(sf::Color::Black);
            Border.setOutlineThickness(1.f);
            Border.setPosition(801.f, 81.f);

            BaseSprite.setPosition(Border.getPosition().x + 19, Border.getPosition().y + 40);
            TurretSprite.setPosition(BaseSprite.getPosition().x + 15.5f, BaseSprite.getPosition().y - 21.25f);

        }

    public:
        //Setter Methods
        void setBasePosition(float x, float y)
        {
            BaseSprite.setPosition(x, y);
        }

        void setTurretPosition(float x, float y)
        {
            TurretSprite.setPosition(x, y);
        }

        void resetPosition()
        {
            BaseSprite.setPosition(Border.getPosition().x + 19, Border.getPosition().y + 40);
            TurretSprite.setPosition(BaseSprite.getPosition().x + 15.5f, BaseSprite.getPosition().y - 21.25f);
        }
        //Getter Methods
        sf::Vector2f getBasePosition()
        {
            return BaseSprite.getPosition();
        }

        sf::Vector2f getTurretPosition()
        {
            return TurretSprite.getPosition();
        }

        sf::Sprite getBaseSprite()
        {
            return BaseSprite;
        }

        sf::Sprite getTurretSprite()
        {
            return TurretSprite;
        }

        sf::RectangleShape getBorder(int money)
        {
            if(money < cost)
            {
                Border.setFillColor(sf::Color::Red);
            }
            else
            {
                Border.setFillColor(sf::Color::White);
            }
            return Border;
        }

        int getCost()
        {
            return cost;
        }

        std::string getInfo()
        {
            return info;
        }

        sf::Time getReloadTime()
        {
            return reloadTime;
        }

        sf::Sprite getBullet()
        {
            return BulletSprite;
        }

        int getDamage()
        {
            return damage;
        }

        sf::Texture BaseTexture, TurretTexture, BulletTexture;
    private:
        sf::Sprite BaseSprite, TurretSprite, BulletSprite;
        sf::RectangleShape Border;
        int cost = 2500;
        int damage = 10;
        sf::Time reloadTime = sf::seconds(0.5);
        std::string info = "Weapon 1\nReload: 0.5s\nCost: $2500\nDamage: 10";
```

Consider changing items related to Weapon 1, whose class is shown above. The code may seem difficult but it actually simplifies maintenance. To change the cost or damage of Weapon 1, I simiply need to change the corresponding variables. These values will hold true for all Weapon 1's when they are added, and to change these values, we only had to make one change. This is good robust code. The same can be said for the reload times.

For the sprite image, the image can easily be changed so that the enemy has a different texture, but the way it has been programmed means it has to be 40 x 40 pixels. This can be fixed but will require a lot of changes throughout the game. However, this is done on purpose as a 40x40 image is best suited to the game and ensures that the game does not look ugly as may be the case with smaller/larger images.

So the combination of differnet modules for different functions and the use of classes means that for small changes, the game is fairly easy to maintain. Bigger changes of course require code addition so may require more coding but as each module is self-contained, it shouldn't require any major overhauls of the code.

Limitations of the game

The game has been finished but it is not a perfect game. There are still some areas which it could improve on. The biggest area the game needs to be improve on is memory usage. Right now, there are not a lot of heavy textures being used by the game. However, for every object created, a separate texture is being stored by the code in memory to ensure it renders correctly. For a few sprites, this is fine as it doesn't take up that much memory, but stuttering becomes an issue as the game progresses making it not function as fully expected.

The way to overcome this would be to create a resource manager which is quite difficult. However, if say we have 100 Enemies on the Screen with 20 Weapons to fight them, that is 120 Textures to store. However, there are only really going to be at most 6 different kinds of textures being used. Therefore, a resource manager would allow me to only load those 6 textures into memory and assign each sprite the relevant texture to ensure limited memory use. This would make my game run far more smoother than it currently is.

In addition, some may argue the game isn't fully balanced yet and may even escalate far too quickly. This is because one mission you can have about 20 enemies and the next you can have 50 enemies. This is largely down to the random nature of the code – the random selection of enemies means there is limited control on what enemies appear. This could be fixed in two ways. One is to tweak the values so that the game does not accelerate as quickly as it currently does. This is easy to change but difficult to get right. Another option is to have percentages for the type of enemy. For example, I can have it so that only 10% of enemies are type 1.

Future Developments

My game is now “finished” and yet far from finished. Due to the limitations I pointed out in the Analysis Stage, my game is nowhere near as good as it could be if an entire team was put to the task. Therefore, there are various additions I would make to the game:

1) Weapon Upgradability – As I mentioned, this was removed during the development stage as it was not considered necessary and I was tight on time to complete the game. However, this would be a brilliant addition to the game as it would make it more strategic and was a highly anticipated feature of the game by my stakeholders. This is essential to the game as it allows even more interaction during gameplay. Right now, the only possible event during simulation is the use of powerup which even then can only be used once, so by introducing Weapon Upgradability, I allow there to be even more enjoyable features in the game making it a lot more fun.
2) Sound Effects – This was one of the opitional additions which would have made the game more engaging, as sound effects make it more realistic and music can make the game more entertaining. In addition, music can really liven up a game and make users want to play a game for longer as well as associating the music with the game alone. Therefore, sound effects will enhance gameplay.
3) Hotkeys – Pressing the 1 key should allow the user to drag Weapon 1. This just makes it easier for gamers who really enjoy the game and wish to get fully involved with it. As seen in my research back in the Analysis Stage, hotkeys were often used by Tower Defence games and they were effective in providing the best gamers the most comfortable experience possible.
4) More than one map – Right now, the game keeps running on the same map which will eventually become dull. Later on, it would have been useful to have more than one map that the user could play on and would make the game a lot more fun than it is right now.

I also asked my users this question – what would you add to the game. There were some good suggestions which were as follows:

5) Health bar values – it’s all good having health bars, but having a number to go with it would be useful as each enemy has different healths wheras the health bars only show percentages of health. This allows the user to make more informed decisions before pressing the powerup button and also makes it much more easier to read the health of enemies.
6) Resizable window – right now if the game goes full screen, it may not work properly or it won’t look as great. Resizability would be a great feature to make the game more personal as not all displays are of the same size. Just because the game looks fine on my laptop, does not mean it will work on all computers so that is something that needs to be considered.

Therefore, although the game is complete, there are plenty of things I could have done to make the game even better. All in all, the process of making this game has been deeply enriching to me and I know for a fact that not only have I enjoyed the journey, but my stakeholders were also satisfied with my input. With more time, I may have been able to make my game even better and add even more features.