**Evaluation  
Ghusharib Chohan 605**

**Testing the robustness of the game**

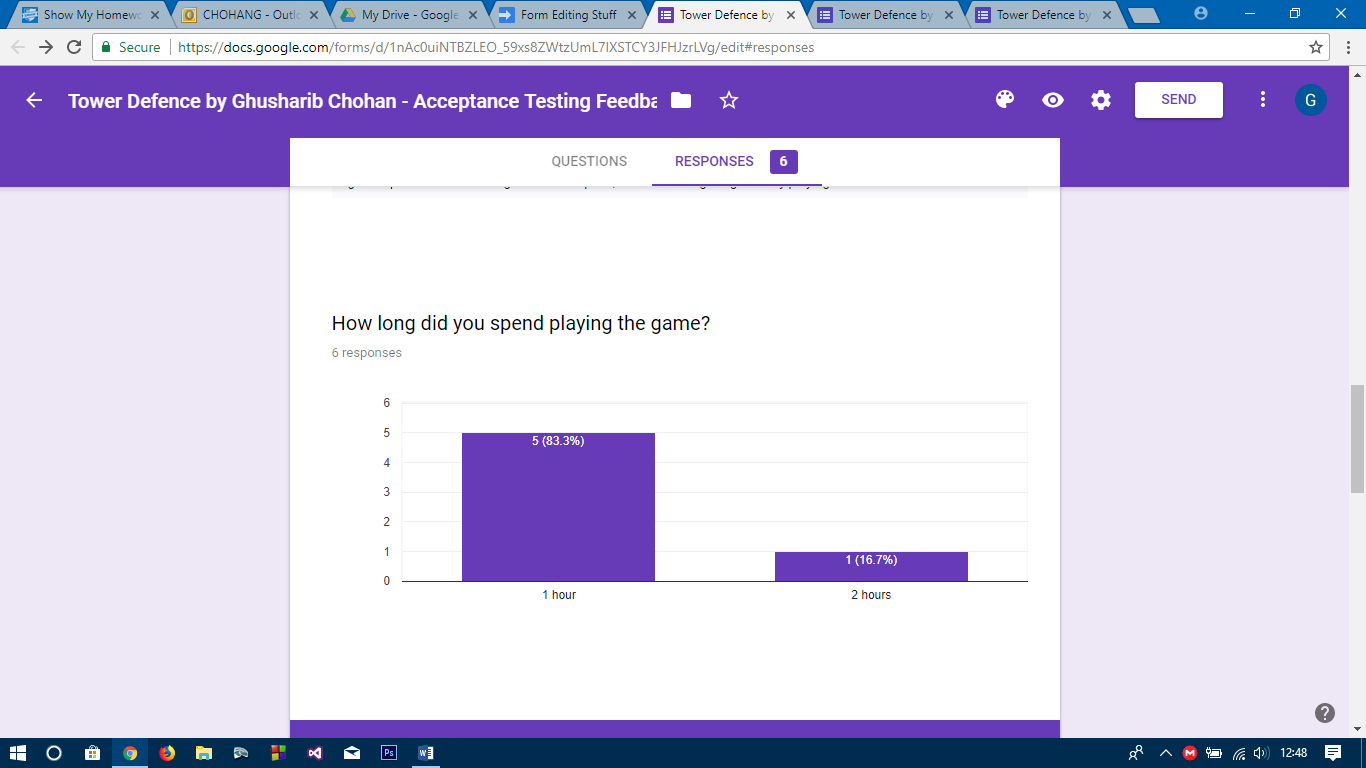
Back in the development stage, I had during development testing which followed my test plan from the design stage. I have then carried out post-development testing and user testing as can be seen in the pages before.

**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:

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.
2. The game should run fairly smoothly.
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 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:



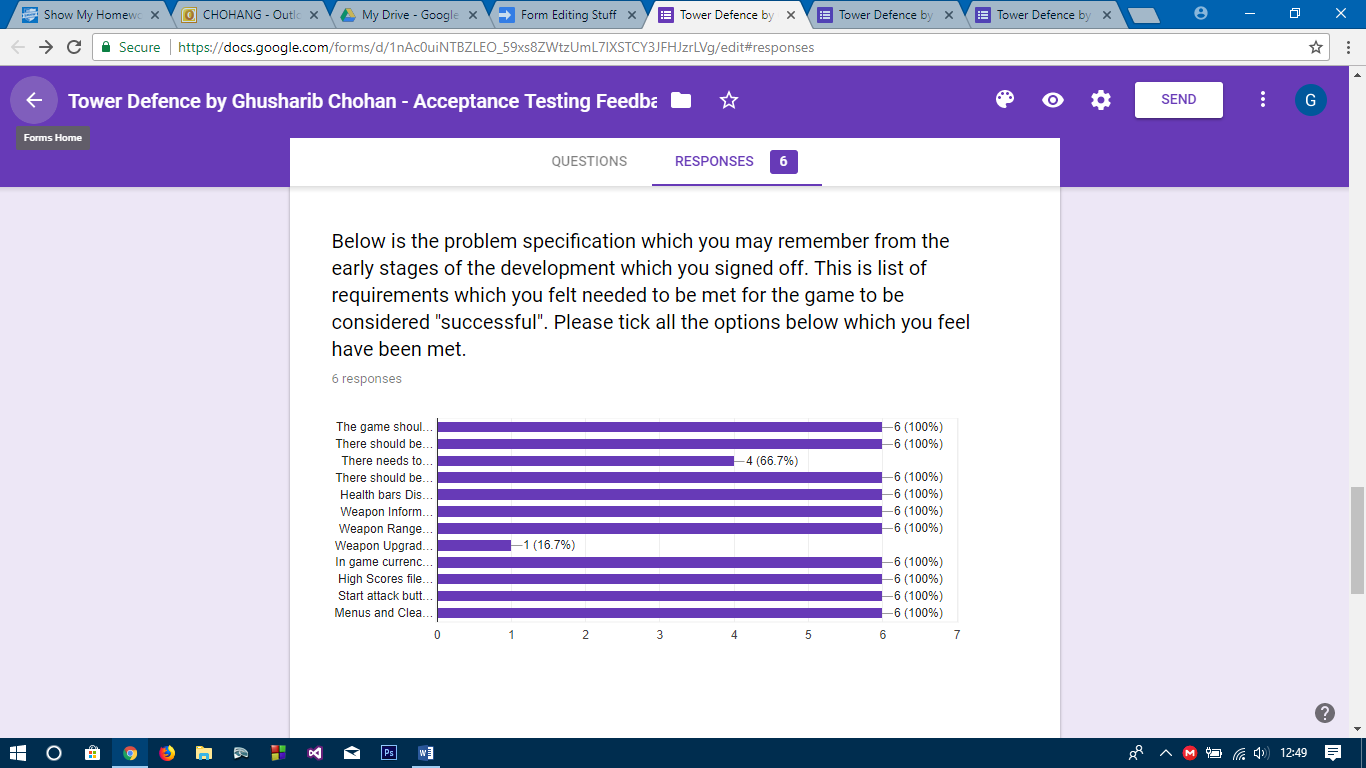
5 out of 6 of my users played for 1 hour and one of my users even played for 2 hours! This has breached my expectations of half an hour as 100% of my users played for longer than half an hour. This suggests that the game was addictive and that they genuinely found it entertaining albeit there were a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

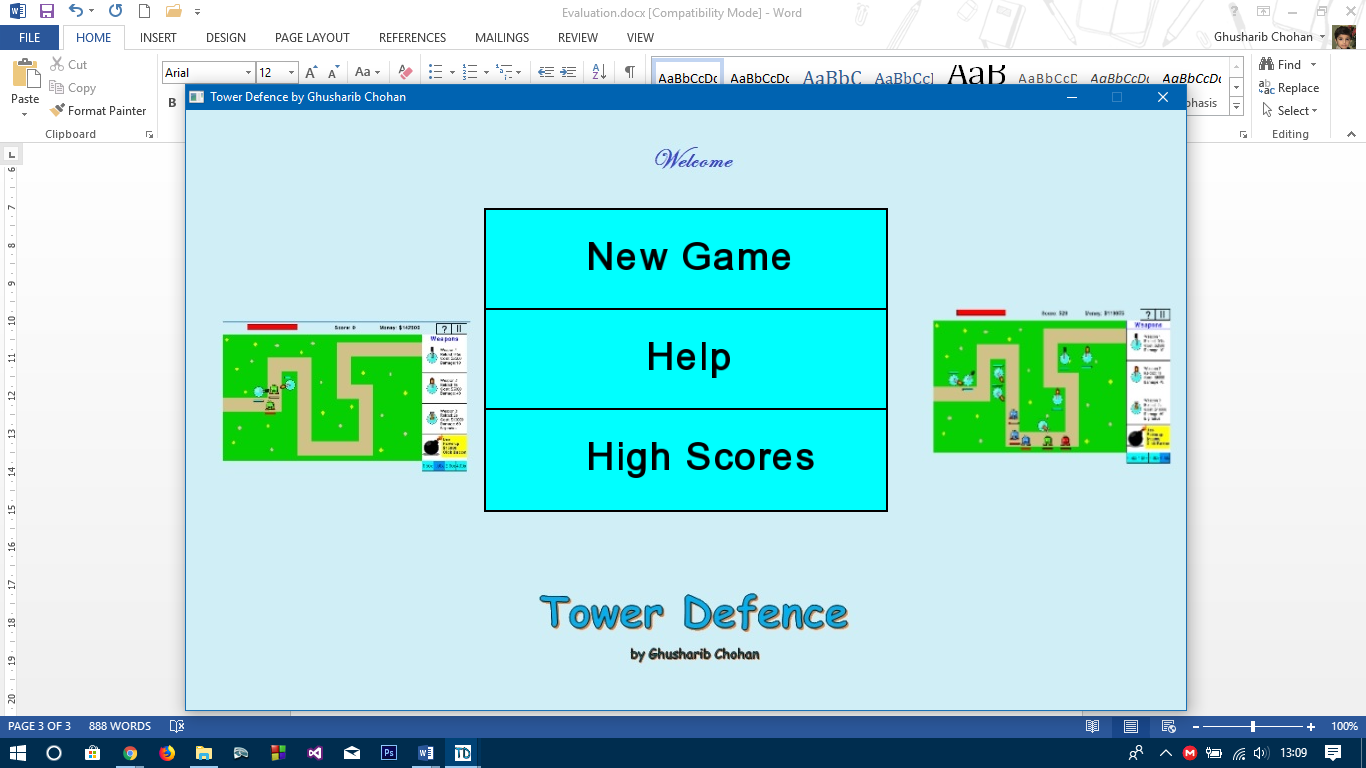
For my game, I decided on some usability features that needed to be implemented back in the design stage. These were as follows:

* 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**

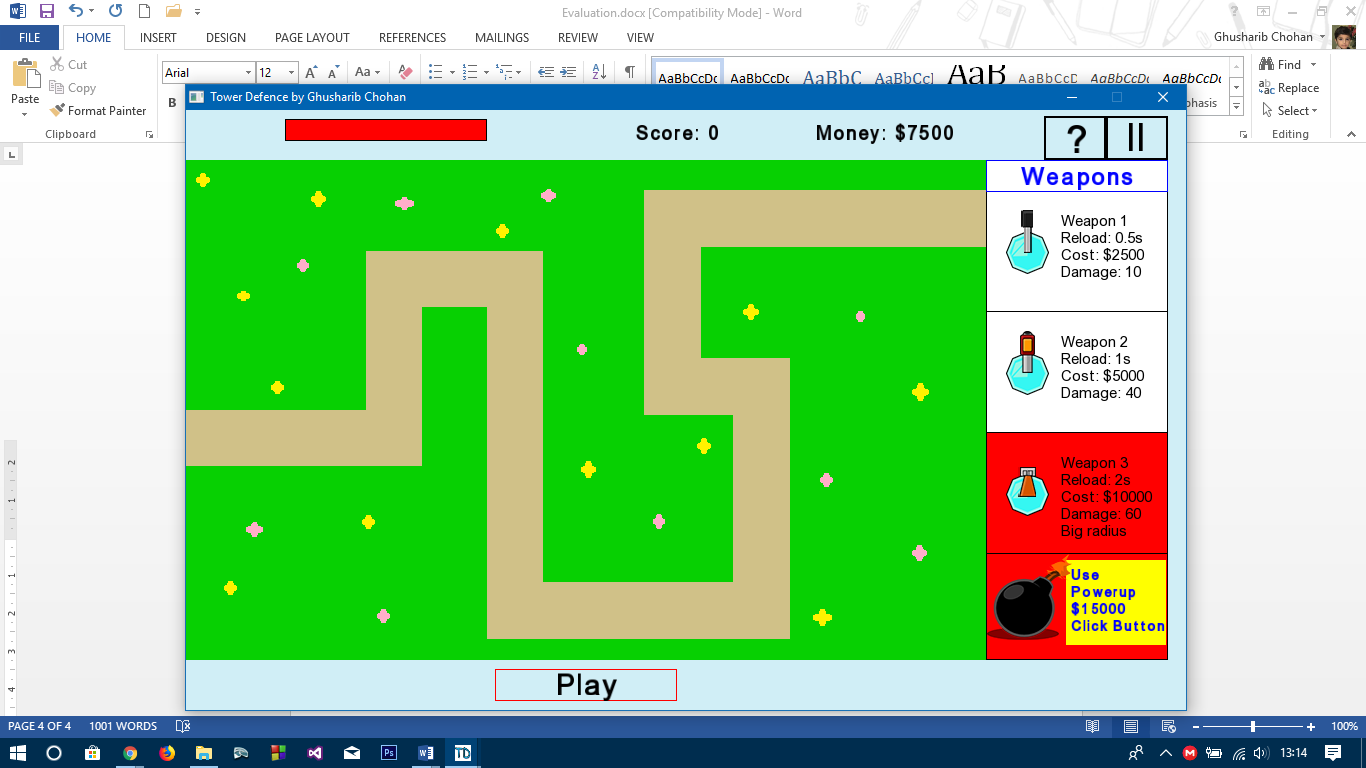
The help screen was a feature that was 100% required. My users often did not know how to play the game and by having a help screen available, they were easily able to understand how the game works and how they should play it. This also needed to be intuitively placed. Therefore, this was how it was displayed:

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.

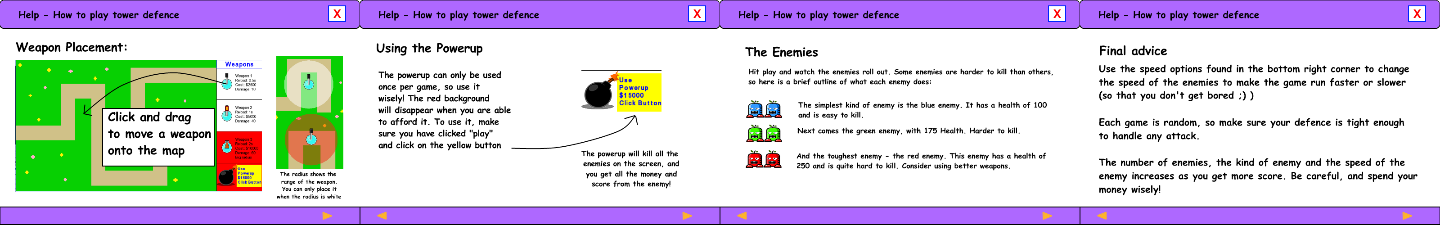
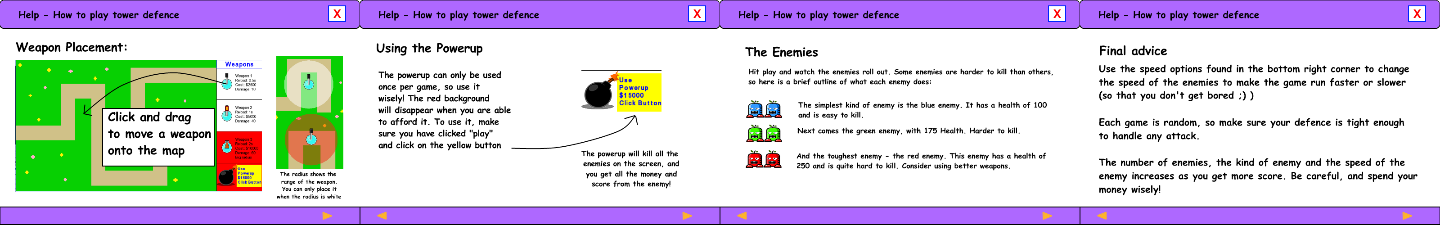
 The “Help” Button appears with all the other buttons (New Game and High Scores) so that the user is not left searching for a help button. It is intuitive that they need to click on it if they face any difficulties in the game.

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.

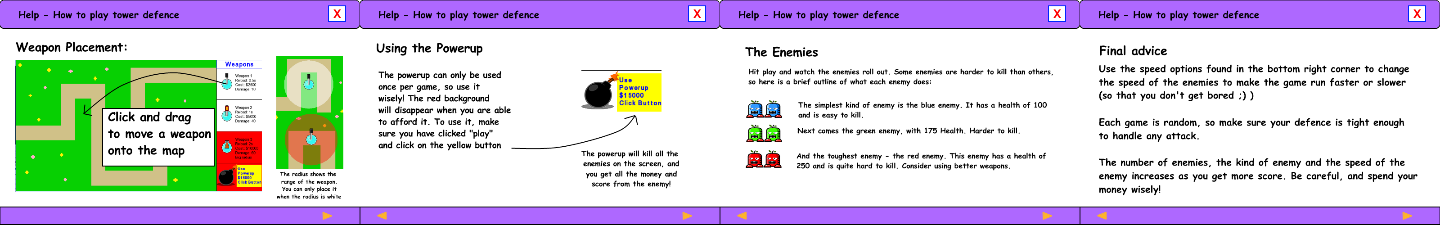
1. When the game is being played, there is a Question Mark in the top right corner. This is as in all applications the standard for short hand help buttons. The user will know to press this button if they require help during the game.

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.

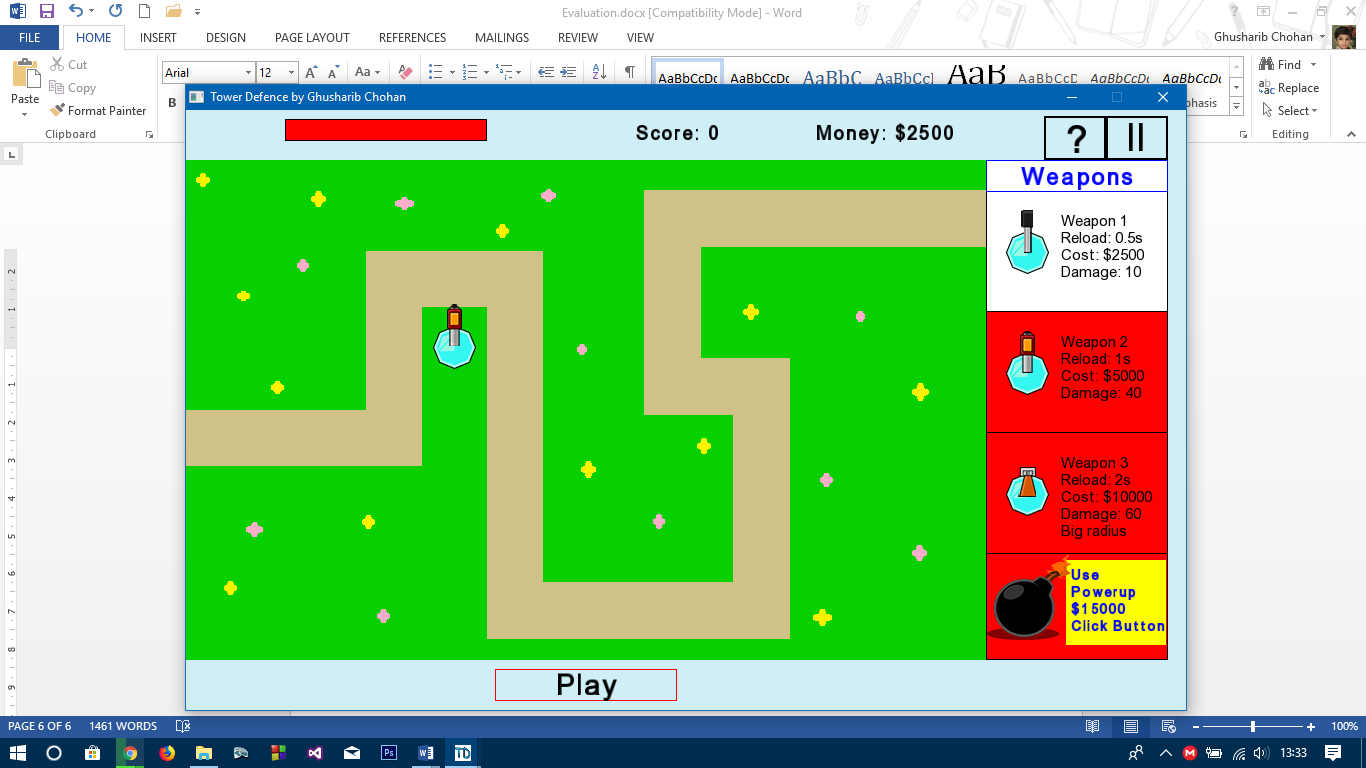
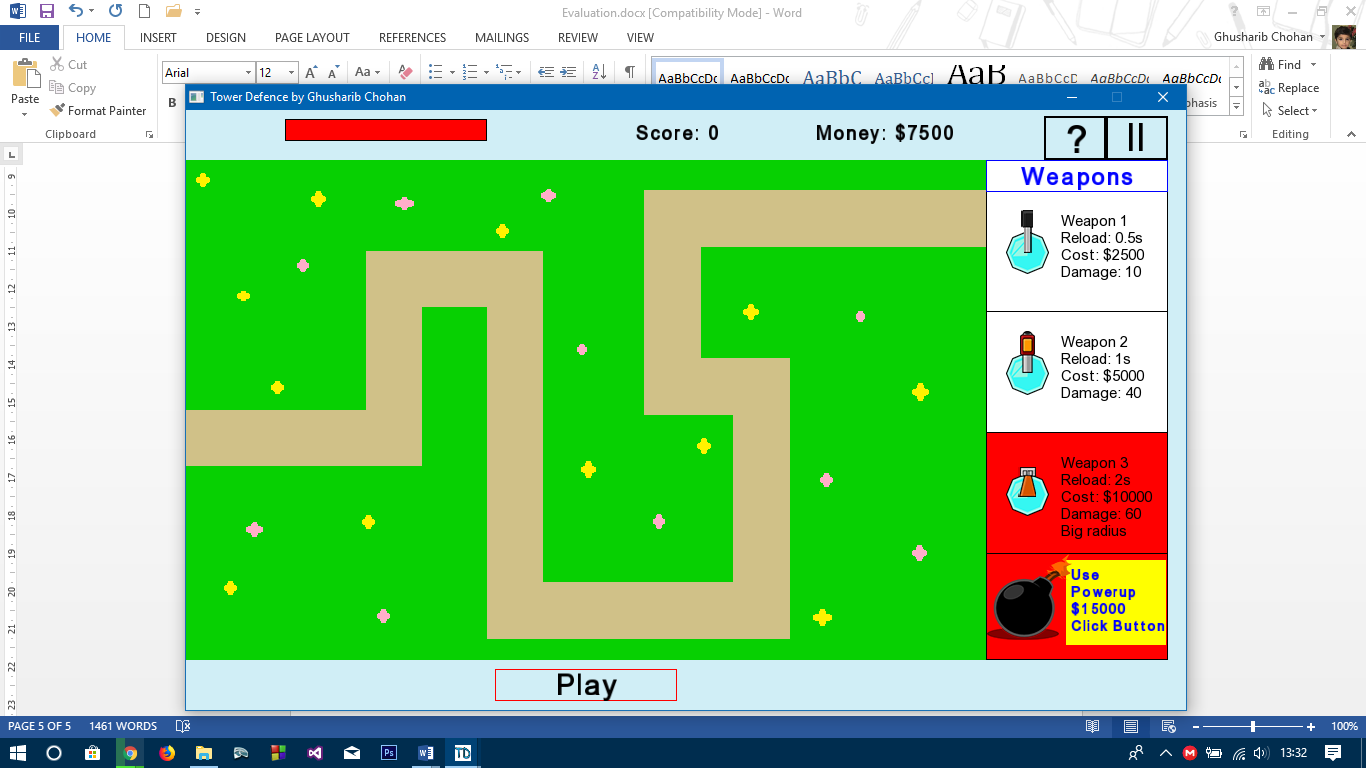
**Feature 2 – Error Messages**

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.

1. 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.

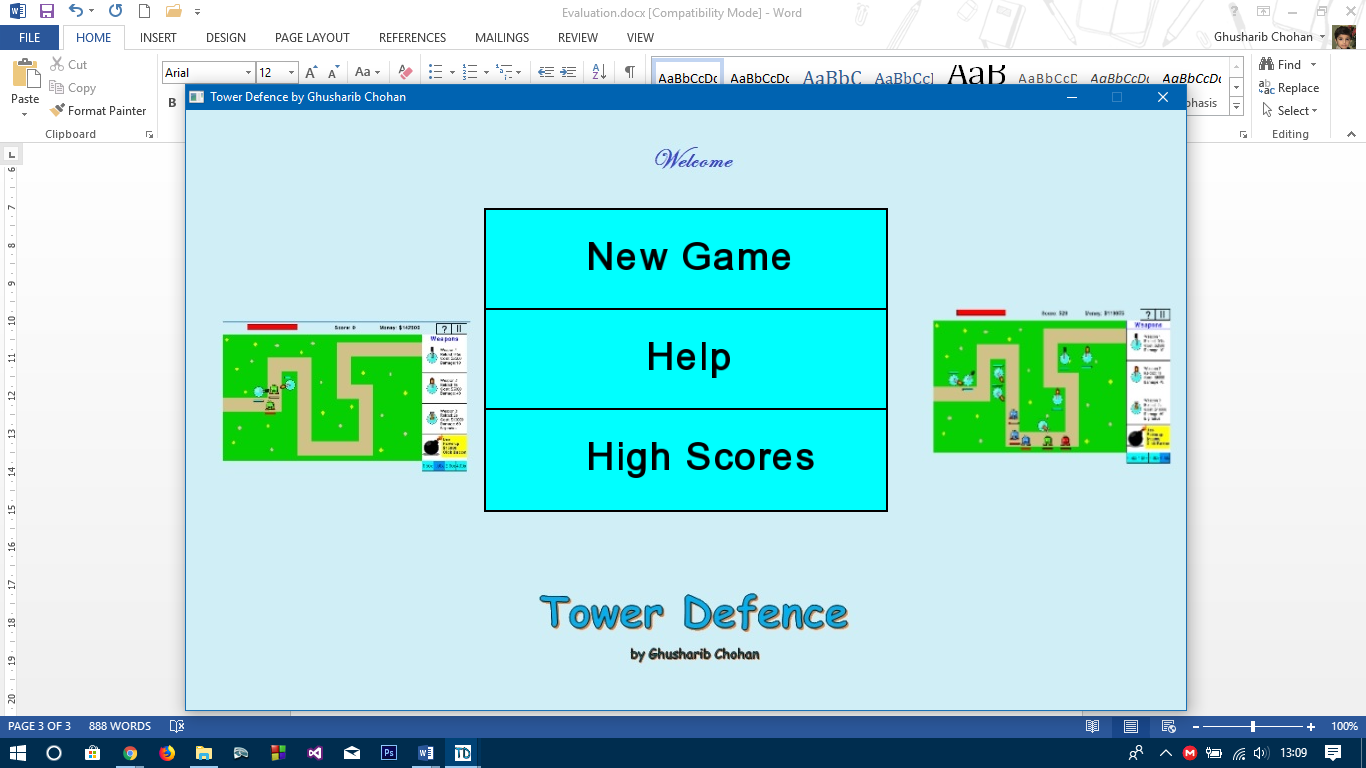


**Feature 3 – Speed Buttons**

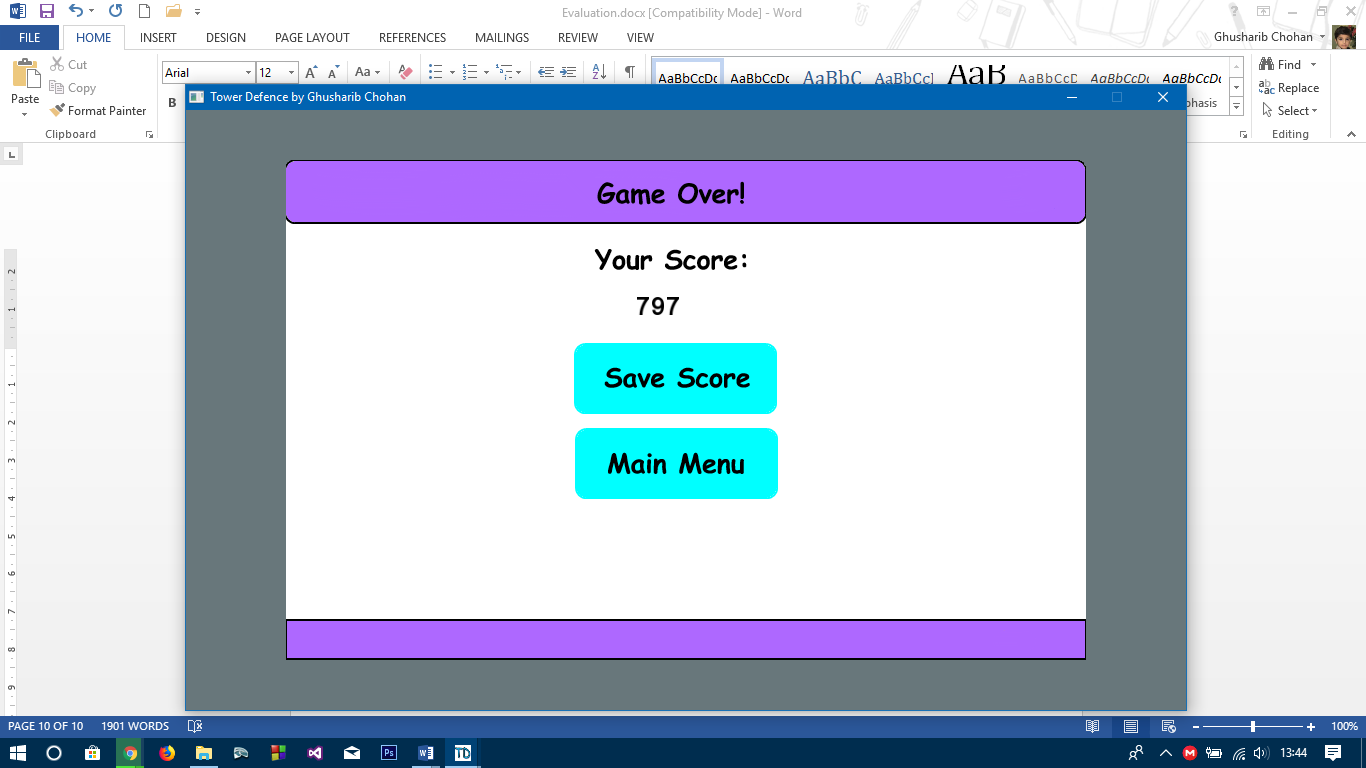
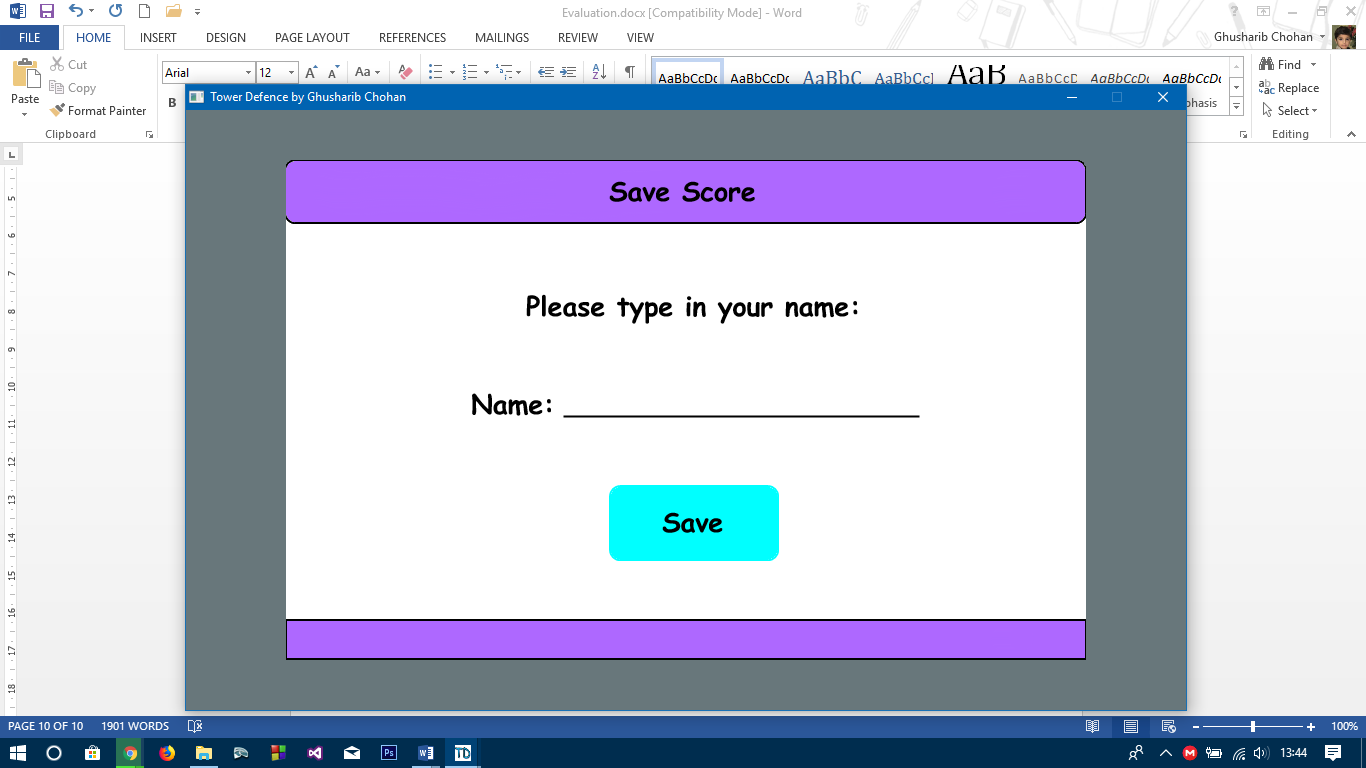
These were placed in the bottom right corner which is an intuitive place for the speed buttons:

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:

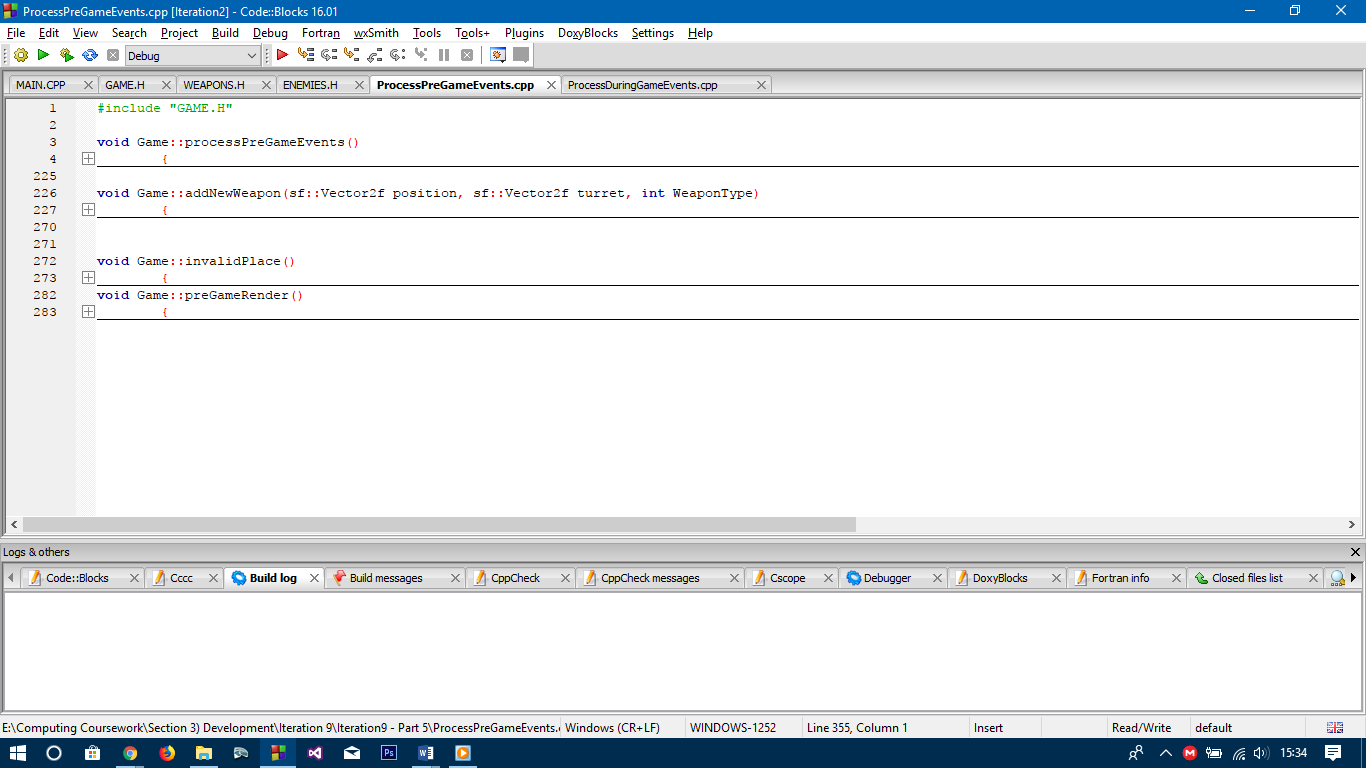


Therefore, all my usability features have been successfully implemented and make the game easy to use as required.

**Maintenance of the game**

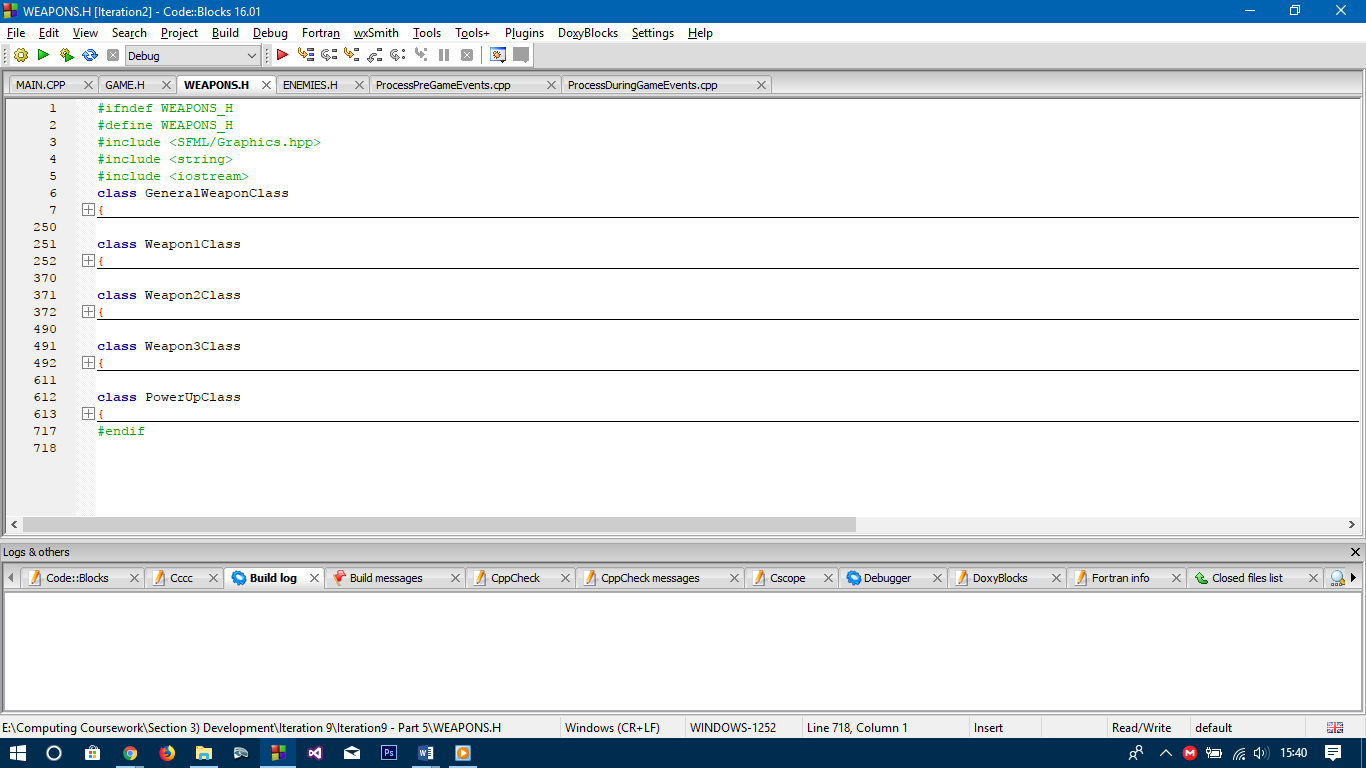
A measure of how well I coded my game is how much change I need to make to the code when adding to it and fixing bugs. If the program has been well coded, there shouldn’t be too much requirement to change major parts of the code.

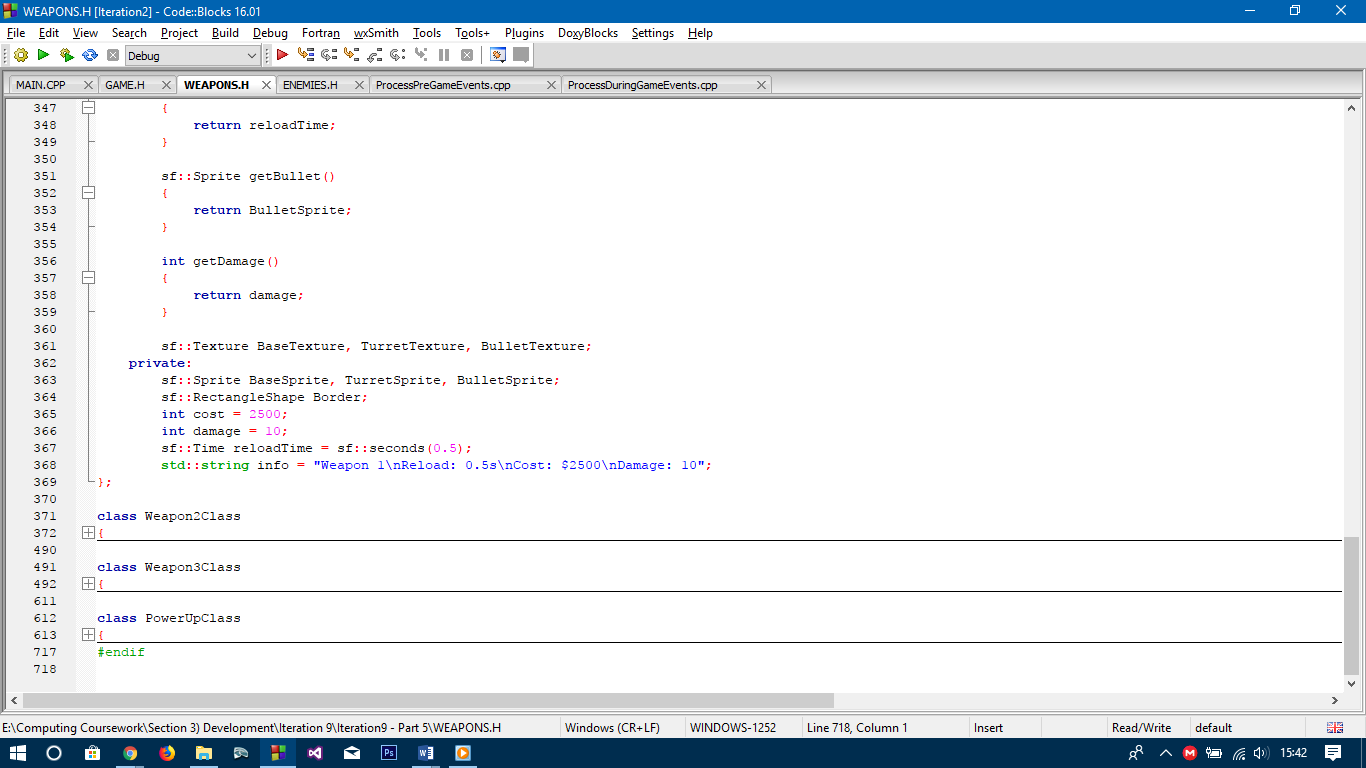
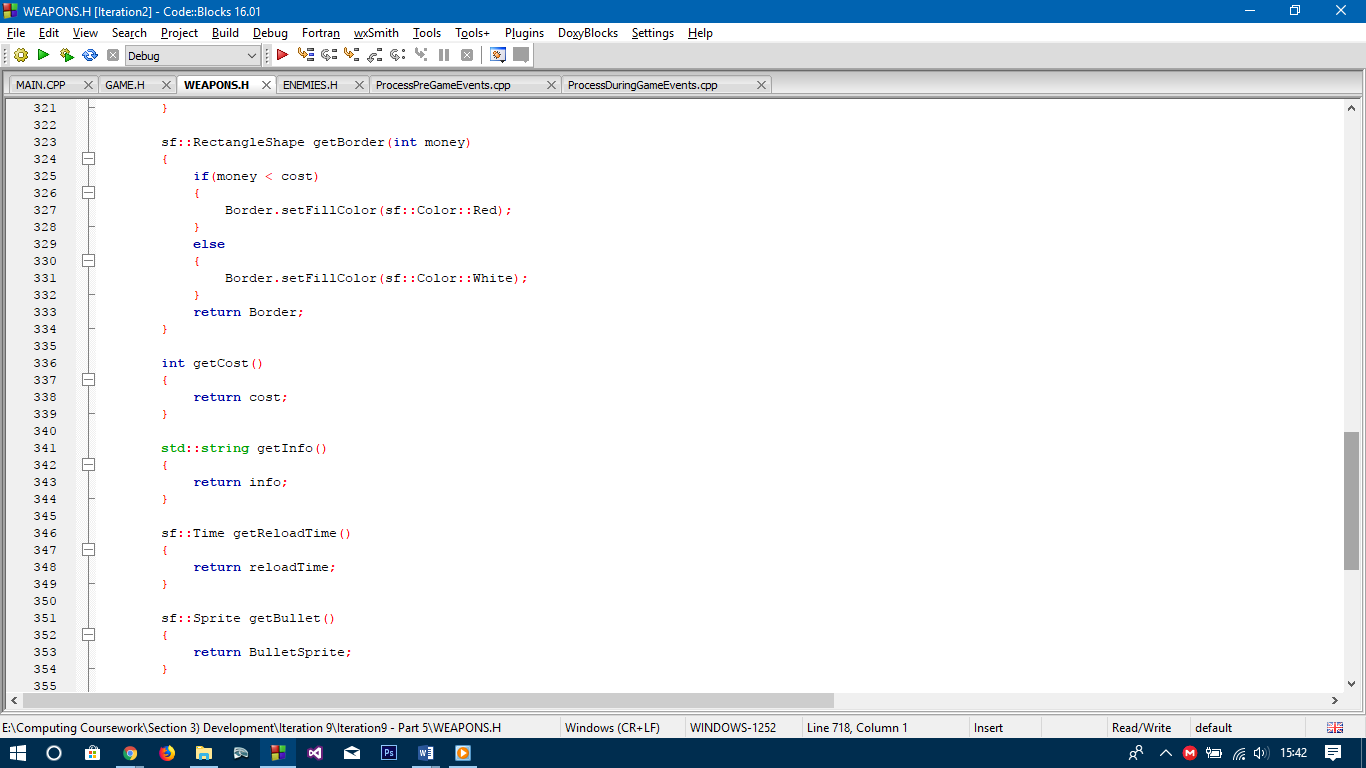
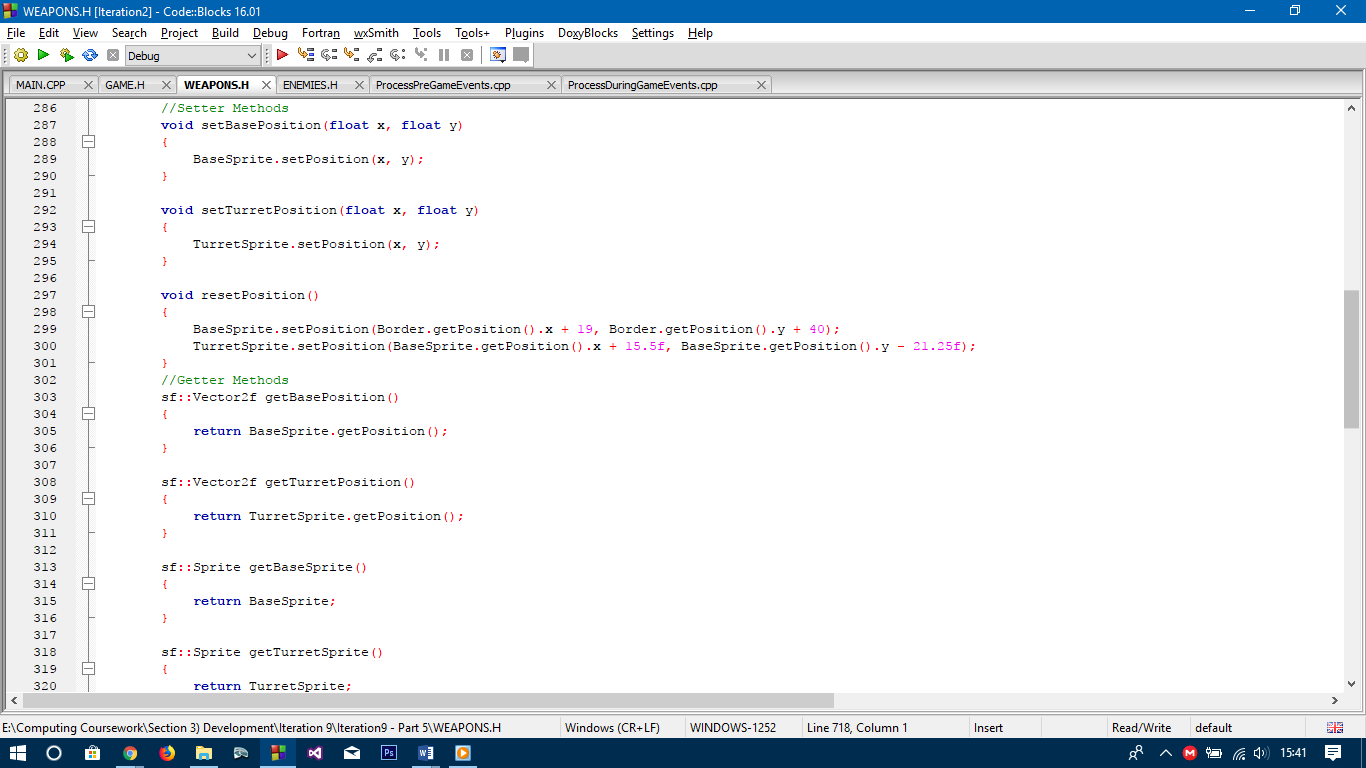
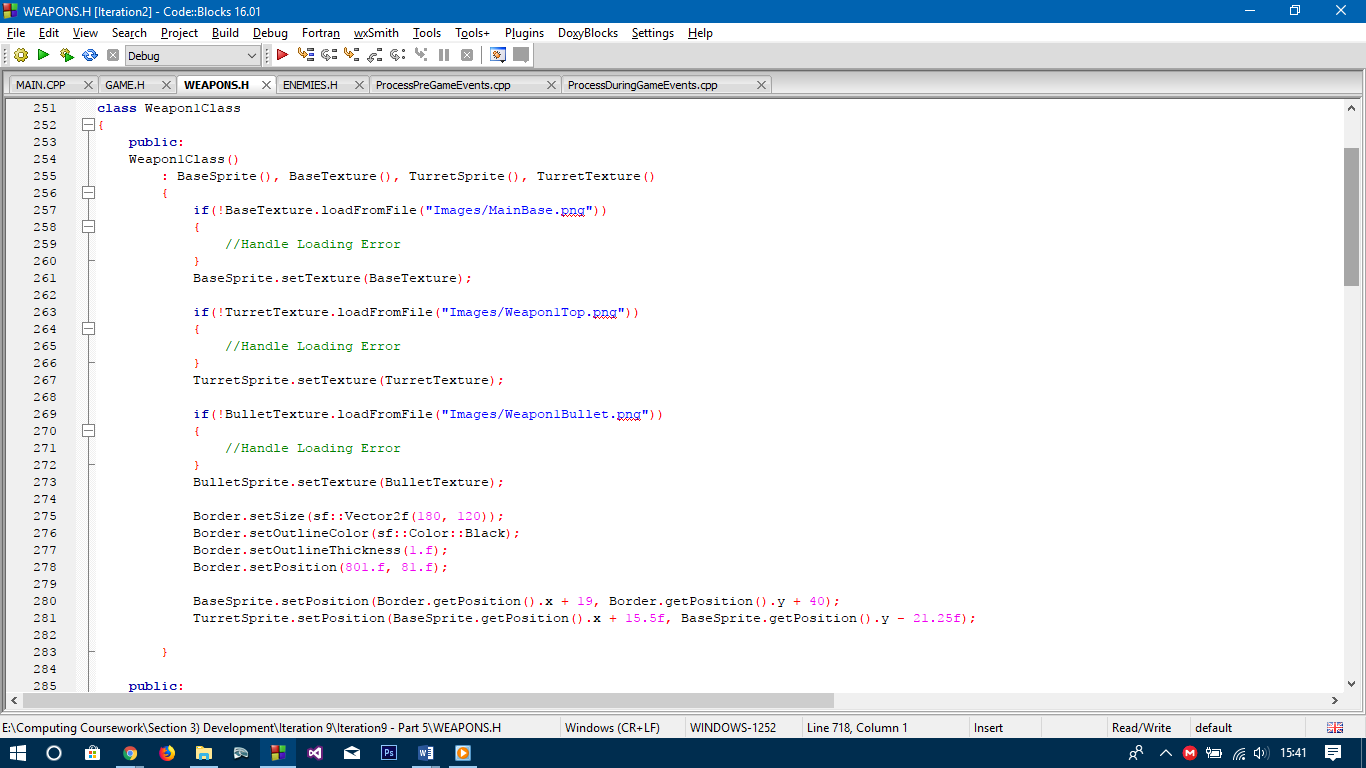
The biggest benefit of my code is that it is a modular design. My code has been broken down into smaller parts and each of these parts has its own block of code associated with it. This means that I know which part does what and where I need to insert new code. This gives the benefit of simplicity for 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.



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:

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.



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.

**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
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
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 and is therefore something that would have been great to add to the game.
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:

1. 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
2. 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

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.