***2023 Take Home Assignment***

***Design Document***

*Luke Courtney*

**Introduction**

As part of this project we were first tasked with creating a game to fill a set of requirements. We were given an overview of the classes required and their primary functions. These included a gameCharacter class, as well as player and enemy classes that inherited from it, a game class that ran the game, and a main method that ran everything. The second part of the project was to improve the game in some way. I chose to do this in three ways.

**Improvements**

***Game board***

Firstly, I made the decision to draw the map of the game using ascii images. This would make the game significantly easier to play, as the user would no longer have to tell where everything was by reading a list of coordinates. This works as follows:

Text

Description automatically generated

Figure 1 - Drawing the game board

I started by creating two for-loops that loop 10 times, with integers X and Y. These are used to loop through all 100 cells of the 10x10 board *(Line 240 & 242)*. Next, I created a loop that would loop through all of the game characters on a list created when the game was initialized *(Line 247)*. I used this to see if any character in the list was in the cell currently given by the X and Y loops *(Line 252).* From there, I would check what sort of character was in the cell and add the appropriate symbol to a string variable. The player is represented by an ‘o’, enemies by an ‘x’ and the crown by an ‘M’. When the X loop was completed, the line would be printed, and the process would repeat for the next line. The result was a game board like the one shown below.

Background pattern

Description automatically generated with medium confidence

Figure 2 - The game board

***Updated battle system***

Secondly, I improved the combat system given in the brief. It initially worked in way that the object with more health would win a battle. I updated this system to be similar to that of a tabletop board game. Both sides would roll a 20-sided dice and the side with the higher number wins. If the player won, the enemy would be killed, but if the enemy won, the player would take some damage and be able to continue. If the damage exceeded their remaining health, the game would end. This allows the user to take more risks, rather than avoiding the chance of instant loss on contact with an enemy in the older version.

Text

Description automatically generated

Figure 3 - Rolling dice and finding the player

I began by generating two random numbers between 1 and 20 inclusive, simulating two 20-sided dice *(Line 137)* . Next, I got the players position using get methods for the player characters X and Y values *(Line 141)*.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 4 - Finding if the player hits an enemy.

The next step was to figure out if the player shared a space with an enemy. As the player was the first object in the character list created during initialization, this was relatively simple. First, I created a loop through the list, and advanced the iterator by one *(Line 145 – 147)*. This would skip the player in the list and avoid causing them to do battle with themselves. Following this, I checked if the object the iterator pointed to was in the same space as the player *(Line 149).* I then make sure the enemy is alive, and make sure its not the crown instead *(Line 152 & 154)*.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 5 - Comparing dice

The final part to the battle system is comparing the numbers generated as dice rolls by the player and enemy *(Line 163)*. This is simply done by comparing the two values to see which is higher. If the enemy rolls higher, the player takes damage equal to half of the enemies health. If the player rolls higher, the enemy dies.

***Objective***

Finally, I added an objective to the game in the form of a crown that the player must reach in order to win. Represented by an ‘M’, the crown spawns on the other side of the map as the player. As a result, the player must cross a board of moving enemies to reach it.



Figure 6 - Adding the crown

The crown was a simple addition that adds a goal to the game. It was created using a much-simplified version of the enemy object. It inherits from GameCharacter and as such it has health and a speed, but it does not move or battle. It effectively just marks a space for the player to try and reach.

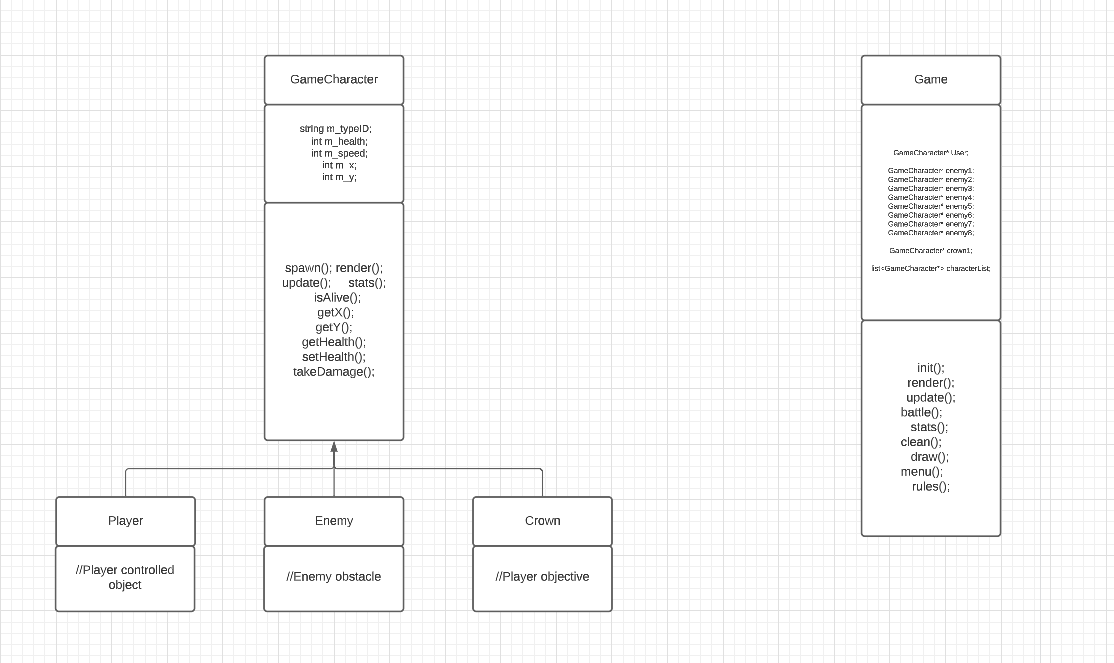
Graphical user interface, text

Description automatically generated

Figure 7 - Win condition

If the player and the crown share a space, the game ends with the message shown above *(Line 122 & 123).*  The game then exits with code 0.

**UML Diagram**



**Lessons learned**

Overall, this project was a useful way of getting familiar with the fundamentals of OOP. It was also a good lesson in working with pointers. Though there were some hurdles initially, with several unfamiliar errors and bugs, I soon became comfortable with working with them.