Cedric Chang
Jay McCarthy
CMPU-203

# Assignment: game2
# Adventure: Legacy

I created my game2 using the Unity game engine. The code was written in C# and testing was primarily done via a rigorous play testing method. Unity allowed me to run the game while watching my variables and gameobjects after every frame in the built-in inspector to determine if and when something went wrong. By running through various scenarios every time something was changed in the game, I was able to find errors either due to conflicting scripts or incorrect implementation. I also ran my tests through multiple iterations of the game during the same run. This method of testing has proven to be an efficient way to demonstrate that the rules of the game are enforced no matter how long the game runs for.

The basic tests that were run each time the Main scene was loaded were player controls and interactions with enemies. First I tested that the player moved correctly and could only move within its boundary. Then I checked the weapon mechanic by shooting a laser and verifying that when it collided with an enemy or boundary, the event took place. This included checking the health values of an enemy before and after collision with a laser. The last of these basic tests was observing the player colliding with enemies. After the collision, the enemy should be destroyed, player health should be decrease by one, and the health UI slider should decrease its value to match that of the player's health.

The next group of tests involved the upgrades players could purchase. Each upgrade would change variables within the gamecontroller which would in turn affect either the player, weapons, enemies, or scoring. When an upgrade was bought, I checked to make sure that the values changed accordingly in the inspector and then loaded the main scene. The values were checked again to see if they persisted. I then tested that each changed variable had the correct effect on the gameplay. For example, when rank and power upgrades were bought, I checked to make sure that each time an enemy was hit by a laser, the correct health value remained for the enemy ship and that the enemy would be destroyed after the required number of hits. The

tests on the upgrades could be run on any upgrade at any time as I could purchase any upgrade at any time using the inspector.

The scoring system required the most attention as the score (or reputation) changed in a different way each time an enemy was destroyed. Throughout all of my other tests, I calculated what the score should update. This meant pausing the game before each enemy was destroyed, doing the calculation for the score by hand, then resuming the game to see if the calculation done by the code matched my own. This had to be done every time an enemy was killed in succession, when the chain was reset to 0, when enemies received an automatic upgrade after 20 seconds, and when the player purchased a new rank upgrade. Though tedious, the tests reinforce my script for calculating the score.

The final portion of testing was done on the linked list I used to store active enemies. Each time the game spawned an enemy and placed the enemy in a list, I had Unity print “Added enemy to list” to the console. Similarly every time an enemy was destroyed, Unity printed “Removed enemy from list”. When the self-destruct button was pressed, Unity also printed out the size of the list. This allowed me to check if the correct number of enemies was destroyed from the self-destruct mechanic by comparing the number of enemies destroyed and the size of the list.