**Software Implementation and Testing Document**

**For**

**Group 4**

Version 1.0

**Authors**:

Andrew F

Bret S

William H

William S

# Programming Languages

For this project we used the C# programming language. C# was chosen for its heavy use within the gaming industry and being one of the languages that was more compatible with the Unity engine. We also felt a higher-level language might be easier to work with since none of our members know anything about unity. Working with low level language such as C++ on top of our unfamiliarity with unity seemed like a heavy load to carry for our very first unity project. C# was mostly used in writing scripts that were imported into our game engine. These scripts were used for character interaction, enemy interaction, and various other aspects.

# Platforms, APIs, Databases, and other technologies used (5 points)

The game engine unity was used along with the C# programming language. No API’s or Databases were used most elements we felt would be better to implement on our own. We aimed for our third iteration to be a to have smoother level transitions, more level variety, and actual death. But as of now none of these additions have called for such technologies as Databases or API’s external to our unity game engine have been needed. Also, we did not want to add in additional technologies when we were not completely sure how said technology might interact with our scripts.

# Execution-based Functional Testing (10 points)

We implemented several new features for this iteration that had to do a lot less with the combat system and more in the way the game looked and played. This was tested thoroughly by repeatedly difficult circumstances such as adding several enemies and testing how they all interacted, slowly developing background objects one by one to determining how they affect the virtual environment. We also tested by using serialized fields for values to determine how fast enemies should move, how high the fire rate for weapons should be, and how high our player could jump during certain situations.

# Execution-based Non-Functional Testing (10 points)

During this iteration there was a bigger push for non-functional testing. It was determined that we needed to test how many enemies could be one the level at once, how high to make objects and how are character need to interact with them, and what it would look like if our game was put under difficult conditions such as too many objects or too many characters could it still push through. These were factors were tested in several different ways by adding various objects and observing how they affect level transitions, by increasing character and enemy count then observing the time duration it took to register hits and collisions, another way testing took place was by increasing environmental objects and gauging how long it took for the initial load in with all various environmental objects.

# Non-Execution-based Testing (10 points)

For Non-Execution-based testing this was spilt among the team was very similar process as what occurred in earlier iterations. Testing such as code inspection that was done by each one of us for all written code that was pushed up to having members verbally walk through and explain their functions/scripts to each other. This cut down on logical mistakes that were plaguing us earlier in the project and thus with more time collected from bug hunting there was more time to implement the features we wanted.