**Software Implementation and Testing Document**

**For**

**Group 7**

Version 3.0

**Authors**:

Sarah Hall

Emily Schall

Justin Williams

Ahmed Alaoui

# Programming Languages (5 points)*.*

C# : using it because we are using Unity.

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

Unity: we chose to work with unity because we were all most familiar with it in regards to game engines. We are using a package within unity called Cinemachine to track the character efficiently and smoothly.

# Execution-based Functional Testing (10 points)

For functional testing, we test our code individually by running the game on Unity, ensuring the results are what we intended as much as possible, and then push to the GitHub repo. For example, player movement was implemented first and tested by ensuring the player was able to walk, jump, and crouch and the correct animations and flipping was done. Player combat and animations and buttons were implemented and tested, as well as combat effects on an idle enemy were tested. Then environmental hazards were added to increase the difficulty of some levels. Lastly UI elements such as health bars and key item pick-ups were implemented to give the player a visual representation of the current state of the game and any existing constraints such as time. The hub environment has been tested to ensure the player can walk around and also use the doors to access the proper level.

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

For non-functional testing, on the player side we tested smooth movement and abilities, and reasonable response time so far, such as walking and testing combat on an idle enemy. We have also tested the look and feel of the objects the player interacts with such as the keys, levers, buttons, and enemies. We test this by adjusting the values of the functions, such as walking speed, attack range, the distance needed to trigger and effect, and more to something that feels fluid and produces consistent results over a few attempts. The variables of these functions were made “public” so that they could be viewed in the Unity Inspector, separate from the code. This allowed us to test the fluidity and response times by changing the values in Unity while the game is actively running.

# Non-Execution-based Testing (10 points)

Some non-execution-based testing that we have done is reviewing functions to ensure they are handling the correct information. This has been done by including Debug.Log statements to check the properties of objects that we are using, such as checking the players current jump status, key pick-ups, and door triggers, and hit conditions with colliders.