Software Implementation and Testing Document

For Group 20

Version 1.0

Ian Guincho

Tony Liu

Christopher Alvarez

Ardonniss Zimero

Programming Languages

C# because it is the universal scripting language used in Unity.

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

Unity, Visual Studio

Execution-based Functional Testing (10 points)

The first functional requirement we had was player movement. In order to test our player

Movement, we built a sample level in Unity with some ground and wall platforms. We then used

our movement script with the main character and tested various movements, animations, turning,

in the box. We added stairs in the room afterwards and tested out the jump code. It is still very

glitchy but we are looking to improve it.

The next functional requirement was level progression. We tested this out by duplicating the sample level we made and by implementing a next level script. We put a cube at the end of the first level and made it invisible. We then attached the next level script to that cube and whenever the character reaches the end of the level, it calls the next scene in the index.

Player obstacles were another functional requirement. We tested this out by using a tile as a trap. We then wrote a trap script and added the script to the tile. Whenever the character touches that tile, or “trap”, the restart function is called in the script, restarting that level. It is the same concept with death by fall, except we have to check the players Y position continuously. If the player’s position is below -1, the restart function should be called in the script. Of course we would have to make all our tiles above this value so that the player does not die on the floor randomly.

The options menu was the final functional requirement. This menu contained our Vsync option, our resolution option, our keybinding option and our volume slider option. In order to test these, we had to run the game and manually click all the buttons to make sure they worked. For the resolutions option, we actually had to build an instance of the game to test the different resolutions.

Execution-based Non-Functional Testing (10 points)

We are going to test for our save state by using different scenes as checkpoints. When the player completes that level, it should save the progress into a json file. This can be easily done by just saving the build index in unity. For example, if the player beats level 4, the save file should have level 5 as the starting point. We have not figured out how to save keybinds yet nor option settings but it should be a similar concept to level saving. Player progress is the more important issue we are working on as of now. This is getting handled by Chris.

5.

Non-Execution-based Testing (10 points)

We all performed non-execution-based testing such as code reviews/inspections/walkthroughs by peer checking. For example, I (Tony) wrote the jump code and helped with the movement code but it was all over the place at first. Not only was it messy code but it did not work perfectly, the reaction time of the movement felt delayed and slow. Ian then found out how to do dynamic key rebinding so we had to revise and change most of the code to fit the keybinds. In addition, he also found other ways to implement movement that felt way more responsive with less lines of code.