Software Implementation and Testing Document

For

Group <3>

Version 1.0

Authors:
Grace Brill
Elias Diez
Jason Graham
Miguel Malayto
Santiago Ruiz

1. Programming Languages (5 points)

The programming language we are using in the project is C#. The reason we are using C# is because the game engine utilizes C# scripts in order to produce different actions and events that the user may want. C# is used in all occurrences for scripts.

We also use the HLSL and Shaderlab code generated by TextMeshPro.

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

TextMeshPro on menus to have the resolution of text scale to different screens and look cleaner than basic Unity text.
Odin (https://odininspector.com/). Not implemented yet but will eventually be used to aggregate variable sliders into singular menus rather than having them spread out throughout the project.
Octave3D (https://assetstore.unity.com/packages/tools/level-design/octave3d-level-design-45021). Unity plugin that we will use for level design as it allows for easier 3D movement of objects through grids and such. Not used yet.
Other Unity Store packs (primarily for visual assets and FX). These will be used throughout the entirety of the project.

3. Execution-based Functional Testing (10 points)

*Describe how/if you performed functional testing for your project (i.e., tested for the **functional requirements** listed in your RD).*

1. Movement System
    a. All movement system implementations were tested by trying all combinations of inputs simultaneously / chained after the other to test for conflicts / glitches that could arise from two separate mechanics conflicting with each other.
    b. They were also tested for input lag by swapping to different keyboards or mice, and on other gaming PCs.
2. First person arm animation
    a. These were tested by doing extremely quick mouse movements combined with fast player movement to test that the overlap between movement due to player rotation and movement due to spatial movement overlapped correctly.
3. General VFX
    a. These are tested by looking at them from all angles that a player could feasibly look at them from, and making sure that they do not break from those angles.
4. Speed Boost Surface
    a. Tested the speed boost surface by creating a prefab called “SpeedArea”. The SpeedArea contains an area hitbox, visual, and animation (denoted as “Speedup”). These visual components were created to indicate where the speedboost region is, which made testing easier.
    b. Testing was conducted by playing the game and running through the speedboost to determine what to make the speed boost interval in the game editor. Through this testing, the speed boost interval was most engaging at 0.1.
    c. Testing also served the purpose of ensuring that the speed boost was unidirectional, so that the player had to enter the speed boost from its starting region, which is indicated by the direction of the animation. This was found to be functional.

5. Slow Down Surface
    a. Tested the slow down surface by creating a prefab called “Slow Area”. The Slow Area contains an area hitbox, visual, and animation (denoted as “Slowdown Particles”). These visual components were created to indicate where the slow down region is, which made testing easier. The slow down region is a pulsating red animation.
    b. Testing was conducted by playing the game and running through the slow down to determine what to make the slow down interval and parameter in the game editor. Through this testing, the slow down interval was most engaging at 0.1 and the parameter at 1.
    c. Testing also served the purpose of ensuring that the slow down was multidirectional. Initially, this code was flawed, but was later resolved and functional.
6. Jump Boost Surface
    a. Tested the jump boost surface by creating a prefab called “SuperJump Area”. The SuperJump Area contains an area hitbox, cube, and animation (denoted as “JumpBoost Animation”). The cube and animation serve the purpose of providing the player with a visual indication of where they can receive a jump boost.
    b. Testing was conducted by playing the game and jumping onto the jump boost surface to determine the minimum velocity for the boost to be applied and how powerful the boost should be. Through testing, the minimum velocity for the jump boost to be applied is -10 and the strength is 50 (using the impulse another team member designed).
7. Managing Player Impulses
    a. Managing the amount of times that the player could spam the spacebar and receive an impulse was necessary to prevent them from being able to, in essence, fly.
    b. This was solved by creating a cool down period, which can be edited in the game editor.
    c. The cool down period was determined to be 0.75 through in-game testing to see what felt right; meaning that the player still could do parkour without having too many unfair jumping abilities.
8. Dialogue
    a. Dialogue was tested through in-game testing and experimentation.
    b. Through testing, there was found to be a minor error with the button which lets the user choose a response, which always shows up. The button is only meant to show up if the user has a response option. This will be resolved during increment 2.
9. Pause menu
    a. Tested the pause menu in the test scene.
    b. Built the game as an application to test the pause menu buttons.
10. Settings Menu
    a. Build Unity game to test if resolution, volume, graphics, and toggle fullscreen worked.
11. Main menu
    a. Tested the main menu by loading the next scene with the new game button and designating a test level as the next scene in the project build settings.
    b. Built the game as an application to test settings menu and quit buttons.
12. Enemy
    a. Tested via the unity engine, through gameplay. I added obstacles for the enemy to fly around and over in order to track the enemy. I also added different patrol points and as many test cases as possible in order to make sure that the enemy was doing the exact things that we wanted it to do.
    b. Also added visuals which were easy to test and manipulate via the unity engine.

    c. Finally, I had a team member review my code to make sure that it was functioning the way that the group wanted it to and made sure it was something that looked towards the final end goal.

13. Minimap
    a. Tested by moving around the entire map and adjusting layers based on object height. Player icon is prioritized and is set to the top layer.
    b. Camera zoom was also adjusted to give the player an acceptable view of the map layout.
    c. Minimap icon positions were adjusted to accurately represent the map layout.
14. Goal indicator
    a. Tested by moving the player camera through all possible angles, ensuring that the indicator can only be seen at the position of the goal.
    b. Distance tracker was tested by moving towards and away from the goal.

4. Execution-based Non-Functional Testing (10 points)

*Describe how/if you performed non-functional testing for your project (i.e., tested for the **non-functional requirements** listed in your RD).*

Thus far, the project has been tested by sending friends with gaming PCs executables to test if it runs wells, thus far, it has run at extremely high framerates.

5. Non-Execution-based Testing (10 points)

Describe how/if you performed non-execution-based testing (such as code reviews/inspections/walkthroughs).

Weekly discord meetings to review any conflicts that the team faced and ways to help fix the issue. We reviewed each other's work visually through the Unity game engine and suggested ways to improve the project. Just anything to help each other was our form of a demo/inspection.