Software Implementation and Testing Document

For

Group 9

Version 1.0

Authors:
Alice Bishop
Andrew Eikman
Christopher SanGiovanni
Jon Guzman
Mathew Paravila Jose

1. Programming Languages

.

- Language(s) used:
    - C#
        - Used where?
            - All scripts in the folder Assets > Scripts are written in C#. This includes, but is not limited to, scripts such as those that handle player movement, item collection, UI elements such as the main menu and scoring.
        - Reasoning?
            - Convenience: The Unity development team has included many libraries and documentation that can immediately be used very easily to implement game features or handle aspects of the game such as scene management or game physics. This is because these C# libraries were specifically made by the Unity developers for use with the Unity Editor. Additionally, Unity has many shorthand features in the editor made for the convenience of those who want to implement C# scripts into their game, such as the ability to drag and drop scripts onto game components, or the ability to create a new C# script through the drop-down menu accessed via right-click.

2. Platforms, APIs, Databases, and other technologies used

- Our main platform for creating this project is the Unity Editor, version 2019.4.13f1
- On this platform, we used many C# APIs provided to help with easy management of the game such as:
    - UnityEngine library
    - UnityEngine.UI library
    - UnityEngine.SceneManagement library
    - System library
- Asset packs such as
    - “Spaceman” https://assetstore.unity.com/packages/2d/characters/spaceman-180967
    - “Pixel Font - Thaleah” https://assetstore.unity.com/packages/2d/fonts/free-pixel-font-thaleah-140059
- Used Pixel Studio, a free pixel art editor, in order to edit sprite sheets to create different character skins.

3. Execution-based Functional Testing

- PlayerMovement
    - First, the code for the player moving back and forth was created, then tested by running the game to see if the character was responding appropriately to keyboard input and moving in the correct manner
    - Afterwards, the code for jumping was created, then tested repeatedly by running the game and testing for a response to keyboard input.
    - The Script for wall jumping and sliding are already there but it hasn’t been implemented into the game player yet.
    - At this point, in the testing phase, the movement did not have any animation to go with it. Proper movement in the right directions was the only concern, and the enum in this script used to interface with the animator was not added until movement was perfected.
- Player Life

    - Testing by repeatedly running the character into traps until the appropriate behavior was obtained. It was through this method that the need for a pause in between death and reloading was found, as the instant transition made it so a death animation was not visible at all.
- User Interface
    - Testing for the main menu interface and ending interface was done by manually completing the game to make sure that it transitioned naturally to and from these interfaces. Booting up the game from the start as a final test to make sure it was working.
    - Testing for the in-game interface was done by moving to several boundaries to see if the in-game UI followed the player properly, as well as if it reloads properly after death or level completion.
- Item Collection
    - For the item collection to be successful, the only prerequisite was that the user be able to walk over an item and have it disappear once overlap between the player bounds and the item bounds was detected.
    - As such, testing involved running the game and making the character run into the items. Once the items began disappearing after the character made contact with them, I could determine that theItem Collection feature was working.

4. Execution-based Non-Functional Testing

- Code Readability + Documentation
    - Group review of code that is working but difficult to read or understand was done to determine where code could be rewritten to be more understandable and reusable in the future. This was done as an iterative process.
- File Organization
    - Existing tools already built into Git and Unity were used to create an easily understandable and optimized structure for our project build. Review of the file structure was mainly done when member-made project files were created to ensure that an easily understandable file structure was preserved.
    - This was done by deciding to find a particular component that the member worked on, then attempt to find that component using only the structure and naming scheme provided by the contributing member to navigate through the files.
- Game performance
    - By running the game and using the Unity Editor tools to check the game stats as it was running, I was able to determine that, even after playing through the entirety of the levels, as well as the main and end menus, that no significant drop in framerate or performance could be detected.

5. Non-Execution-based Testing

- This was done mostly in the form of code review
- When a new C# script or modification of an existing C# script is introduced via a pull request on GitHub, a review of the code included is done to ensure that it semantically makes sense and will do what it is intended to do, assuming that compilation testing has already been done and no compilation errors exist.
- In addition to logical review of code, code is reviewed to ensure readability and reusability, as whenever a member works on a script, the work they do will affect any other member that must use or edit that same script.