Software Implementation and Testing Document

For

Group <19>

Version 1.0

Authors:
Gabriel Clewis
Justin Taing
Brandon Embleton

1. Programming Languages (5 points)

List the programming languages use in your project, where you use them (what components of your project) and your reason for choosing them (whatever that may be).

C# - used in Unity for the entirety of the project
Unity only supports C# as of now so it was our only choice.

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

List all the platforms, APIs, Databases, and any other technologies you use in your project and where you use them (in what components of your project).

Unity is our main engine being used. We use a multitude of packages/libraries provided by Unity in some of the scripting. Some of these packages include:

- **SceneManagement**
- **System.Collections**
- **System.Collections.Management**
- **Text Mesh Pro (for UI text)**

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).*

Functional Battle System – Tested the Fighting scene to properly see if the turns and health changes update properly in both the game and inspector views.
Working Overworld with movement – We implemented player movement and ran the unity scenes to test if player movement worked with the colliders used.

Overworld Scene – Tested the overworld scene to check if the enemy's detection correctly sends the player to the specified scene. Also tested for the rotation of the enemy's collider when the enemy turns in-game.

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).*

Simplicity – We tested the controls and made them easy to understand from a movement and fight-scene gameplay perspective.
Dynamic – We made the fight scene dynamic in that unit's can be added in and the turn system from BattleManager will react as needed. We also made the overworld scene dynamic in that it will send the player automatically to the fight scene.

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

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

Most of our code testing was dependent on actually running the Unity scenes and seeing if they functioned properly. Though, we checked each other's code to make sure it would run efficiently and correctly. We checked each other's implementation along with checking the code that goes along with it.