# Software Implementation and Testing Document

# For

# Group 6

Version 3.0

**Authors**:

Riley Arunakul

Gabriella Estevez

Braden Ford

Kevin Foughty

Mattan Pelah

## 1. Programming Languages (5 points)

C# scripts for editing GameObjects in the Unity Game Engine editor. We're using C# as it's the programming language supported by Unity.

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

Since we are developing a video game from scratch we currently aren't using any software platforms or databases for our project. It's likely in the future we may use a database for player data or for a leaderboard. However, we are using an API called, "Unity Scripting API." Unity Scripting API is why we are using C# for our project as it allows direct integration with GameObjects in Unity. This API also contains many subclasses that allow tweaking specific aspects of the game engine to our advantage. Therefore this API has been implemented in every asset of our project thus far.

In order to create a more unified project we used different art packs to decorate our levels. One by "Anokolisa" and another by "PitiT" who provide scene packs for prospective gave developers. The allowed us to create our theme for our project which was vikings.

Anokolisa -> https://anokolisa.itch.io/sidescroller-pixelart-sprites-asset-pack-forest-16x16
PitiT -> https://pitiit.itch.io/free-2d-fantasy-platformer-asset-pack

## 3. Execution-based Functional Testing (10 points)

To test the functional requirements, we created multiple test levels that allowed us to test individual aspects of the game that we worked on. Such as creating a specific level just to test enemies or just to test the movement without outside interference. After this we also had a temporary level where we put all these components together to see how they interact and corrected errors as the appeared. Once our components worked together we made final levels pased on this template.  We were also sure to push our changes in git hub to separate branches so our teammates could look them over and ensure they would not cause conflicts.

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

Most of the non-functional requirements are subjective (i.e. the game should be fun) so we are unable to test those at this stage in development. However, non-functional requirements that were objective (i.e. the game does not crash) were tested the same way the functional requirements were tested.

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

Non-execution base testing was done through the use of the meetings that took place throughout the increment.  As well as our communication during the increments over discord. We created chats for deadlines, links, notes, debuff ideas, enemy ideas, sounds, and a to-do page. This allowed us to stay in contact and up to date with each other so we could have a smoother workflow. We also for a portion of our project worked in Jira to list our tasks and ensure they got completed.