**Software Implementation and Testing Document**

**For**

**Group <Tanx>**

Version 1.0

**Authors**:

Andrew T

Nolan M

Alex G

Blake A

# Programming Languages (5 points)

We are using Javascript across our project with the Phaser interface. We chose this because we are not particularly well versed with game design and this is a developer-friendly tool that allows people with our experience level to build functional projects efficiently and in a straightforward manner. Their documentation and source code is found in github and they provide extensive support for starting the development.

The game menu is being built using html and css for its structure and stylings.

(This change occurred during iteration 1, but we have left documentation here for reference.)

This has changed from our initial plan of using C++’s SDL library, since we could get working quicker with Javascript. The main reason for this switch was the ease of starting up a project and getting the team on the same page. Additionally, the resources published online are a little more helpful for getting to know the system, as they are a little more modern and oriented for a project similar to ours.

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

We are using Phaser’s library to build our game in JavaScript. We chose this because of the simplicity to get it up and running. Instead of focusing on implementing a more difficult and complex system, we can get phaser running quickly and turn our attention to the small details that will make our game stand out.

We will be building the game’s menu and stage selection portions in HTML, so that the user can see a simple format that will plainly display the options moving forward. For simple elements, we have found this to be much more useful, as the design elements are very straightforward and easy to add.

We are using MAMP and XAMPP with Apache servers to host our testing and game design through our localhost. This allows us to have things working in a web browser without any extra costs or difficult steps.

We are sharing our progress on coding in GitHub to streamline working together with pulls and commits. For sharing the different files we are working on (like this file and other requirements), we are using google drive, as it allows simultaneous editing on documents.

We are also using Atom as our IDE because of its simplicity and usefulness across all of our intended uses.

# Execution-based Functional Testing (10 points)

We’ve been testing functional requirements through simple, straightforward tests. Though this smoke test of sorts may not be the best way to find errors, we have been building a basis for the game’s next steps so it’s been sufficient to make the building blocks work. If the code executed as expected, we considered this up to standard because the expectations and goals were pretty concrete and in depth for this portion.

These core functionalities are pretty basic, so they just need to work how expected, rather than be tested against many different cases. Essentially, we are working them until the issues preventing them from functioning correctly are all gone, so that the larger plans will work.

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

The non-functional testing has occurred with the functional testing, simply as eye tests to ensure that the code has been running efficiently, closes properly and works across different computers. Given that non-functional requirements are more easily recognized as secondary qualities of running code, this was sufficient testing practice, as the code either supported/met these standards, or did not when executed.

# Non-Execution-based Testing (10 points)

We have been setting aside time twice a week to meet on Zoom and check each other’s progress and work through the code’s status. Code review and inspections were also performed before our final meeting of the iteration to confirm that everyone’s code lines up, comments were up to standard, and all of the expected work was done. This step was completed later in the process so that all of the code could be seen and checked in one swift step, rather than taking many small, tedious steps each time code was checked in.