**Final Specifications for Gauntlet**

**Andrew Si, Chris Cheung, Nathan Lui, Som Pathak**

1. **Introduction**

Gauntlet is a 2D top down shooter game in which the user navigates a procedurally generated arena fighting a variety of enemies that try to kill the user’s avatar, or player. The user can choose to play as one of six classes that cosmetically change the player’s appearance. Once the user chooses a class, the game starts and the user must use WASD to move and use the mouse to aim and shoot at enemies. As users kill enemies, the player receives points and levels up. Leveling up increases players’ maximum health and damage output and replenishes a small portion of their health. When the player runs out of health, the game ends and the user can choose to play again.

Gauntlet makes use of LibGDX, a Java game development framework, which handles the majority of the graphics and audio backend for the game. Every single class in the game uses at least one of LibGDX’s libraries. Artwork, music, and sound effects in Gauntlet are all taken from the Internet.

The game is displayed using a multitude of Screens. A Screen is a LibGDX interface that represents an application screen and provides methods for displaying and refreshing a screen. Gauntlet has five Screens, which are Splash, MainMenu, CharacterSelect, GameScreen, and GameOver. At any given moment when the application is running, the user is viewing only one of the Screens. The Screens themselves are managed by a class called Malice. Malice extends a LibGDX class called Game that allows the application to have multiple Screens and controls which Screen is being shown to the user.

Splash is the most simple of the Screens. It is basically a credits screen that lists the names of the developers and a list of video games and websites which the developers got graphics and audio content from. After five seconds, the MainMenu Screen replaces the Splash Screen.

MainMenu is the main menu of Gauntlet. It displays two buttons which give users the option to exit the application or play the game. If the user selects the play game button, the CharacterSelect Screen replaces the MainMenu. CharacterSelect shows several buttons corresponding to each class and lets the user choose a class to play as in the game or click the exit button to close the application.

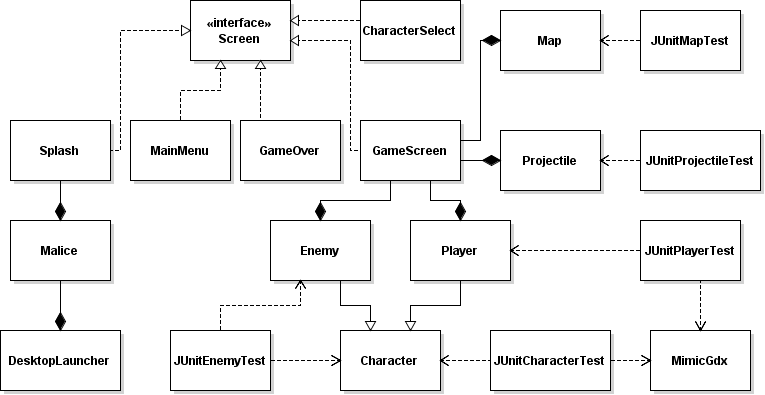
1. **Structural Design**

For our map we used a ***2D Array*** of booleans, because each tile of the map can be represented by the true or false aspect, where true indicates a tile that Characters can walk on whereas false indicates a wall.

In order to store all of the Sprites in the game, two ArrayLists are used. One for the Characters and the other for the Projectiles.

|  |  |
| --- | --- |
| Data | Data Structure |
| Map tiles | boolean[][] array |
| Characters in game | ArrayList<Character> |
| Projectiles in game | ArrayList<Projectiles> |

1. **Object-Oriented Design**



1. **Detailed Design**

A majority of documentation is provided in the API

1. **Detailed Description of Each Class**
   1. **Malice**

The Malice class extends Game and handles all the screens in the Gauntlet game.

* 1. **Splash**

Splash Screen is a Screen that displays the developers of the project and gives credit for all artwork and audio. It also initializes and plays the music. After five seconds, the splash screen is replaced by the main menu.

* 1. **MainMenu**

This screen is the main menu of Gauntlet. It has a background image and allows the user to play or exit the game. The screen also utilizes a ton of LibGDX libraries, including Stage, Skin, and TextButton.

* 1. **CharacterSelect**

The Game Over screen displays the same background image as the Main Menu and allows the user to try again with the same class, choose a different class, or exit the game. The screen also shows users which level they reached and how many points they received. The screen utilizes a ton of LibGDX libraries, including Stage, Skin, and BitmapFont.

* 1. **GameOver**

The Game Over screen displays the same background image as the Main Menu and allows the user to try again with the same class, choose a different class, or exit the game. The screen also shows users which level they reached and how many points they received. The screen utilizes a ton of LibGDX libraries, including Stage, Skin, and BitmapFont.

* 1. **GameScreen**

GameScreen handles gameplay and manages the interaction between the map, characters, and projectiles.

GameScreen does all collision detection between the map walls, characters, and projectiles and determines when characters and projectiles move around the screen. It uses ArrayLists to store all the characters and projectiles currently in the game. GameScreen also calculates the delta x and delta y values for projectile aiming. Furthermore, GameScreen utilizes an Enum to allow for pausing and resuming the game.

* 1. **Map**

Map class displays a map using tiles in a 2D array. Does not use a separate class for tiles, instead it uses a 2D array of boolean to tell between a space and a wall/block. All coordinates are based on a bottom left origin.

* 1. **Projectile**The Projectile class extends Sprite and represents a projectile inside the game. A projectile always moves in a straight line.
  2. **Character**This class represents a sprite in the game with animations that can move and shoot. An object of this type is never actually created during gameplay, only an Enemy or Player.
  3. **Player**This class represents a Player in the game. It takes in input from the keyboard in order to move and input from the mouse in order to shoot.
  4. **Enemy**This class represents an Enemy in the game that will attack the Player when conditions are met. Uses basic AI programming and will move and shoot towards the player in a straight line if within range. Otherwise, it will move around randomly.
  5. **MimicGdx**The MimicGdx class isolates nearly all input from the Gdx class This will allow 2 purposes: - provides the functionality of an options screen when implemented - Allows for better JUnit testing as many methods would be unable to be tested, commented methods can simulate input for testing if implemented.

1. **Testing**

Testing this game consisted of both JUnit testing and actual play testing in order to verify that all of our methods worked as intended. What could not be tested in the JUnit could be tested when playing the game, and what could not be tested when playing could be caught in the JUnits. JUnits could only really test the Map, Character, Enemy, Player, and Projectile classes