**Project Gon Final Specification**

**Description:**

Gon is a 2D top-down view game that brings back the feel of old-school gaming. The main character of the game is Gon, a young warrior on the quest to defeat all of the enemy monsters. On his quest through levels, Gon encounters monsters of various difficulty and must defeat each before moving on to the next level. Gon is designed for gamers of all ages.

**Classes:**

Audio

* AudioPlayer.java

Entity

* AdvancedEnemy.java
* Animation.java
* Enemy.java
* Entity.java
* Player.java

GameState

* GameState.java
* GameStateManager.java
* HelpState.java
* Level[X]State.java
* LevelState.java
* MenuState.java

Main

* GamePanel.java
* ProjectGon.java

Testing

* JUnitTest.java

TileMap

* Background.java
* Tile.java

TileMap.java

**AudioPlayer.java**

**Description:**

Creates a audio player class which plays audio clips

**Function Specifications:**

This class allows audio clips to be loaded in as separate objects. Each clip can be looped, played once, stopped, and closed.

**AdvancedEnemy.java**

**Description:**

This class extends Enemy but has a more advanced and random movement algorithm.

**Function Specifications:**

Unlike Enemy, the AdvancedEnemy is able to move in all 4 directions. The movement is less predictable because it changes direction and attacks at pseudorandom intervals.

**Animation.java**

**Description:**

This class uses an array of images to create an animation with a certain delay between each frame.

**Function Specifications:**

This class allows you to construct an animation by passing it an array of images to cycle through and setting a delay. When update() is called, the class checks if enough time has passed to move onto the next sprite and increments the index if so. Other classes utilize an animation by calling update() at the appropriate time and using the getImage() to draw the appropriate frame in the animation.

**Enemy.java**

**Description:**

This class represents all enemies in the game. This class extends a player class but has automated movement and attacking.

**Function Specifications:**

For movement, position, and being hit, Enemy is just like the Player class. However, movement is automated to move either horizontal or vertical depending on the parameters passed in the constructor. The movement is extremely basic and attacks whenever the direction of the Enemy changes. The Enemy attacks on all four sides and will also attack if it collides with the player. The Enemy class uses one image for the enemy itself and another for the attack image. Each enemy also has a unique block number to identify itself in the array of blocks which are a part of the TileMap for player collision detection.

**Entity.java**

**Description:**

This class is the parent to Player, Enemy, and AdvancedEnemy. This class

holds methods and variables for position, direction, health, speed, width and

height

**Function Specifications:**

The Entity class is the parent to all classes in the game which move around the map, which include the player, enemy, and advanced enemies. The method intersects(Entity e) checks if this entity collides with any other entities in the game using the Rectangle intersects(Rectangle r) method. Entity also handles all of the movement and determines where the next position for the entity will be depending on which direction it is moving in. Lastly, the Entity class provides a method to check for intersection with the blocks on the map to make sure that it does not go past these.

**Player.java**

**Description:**

The Player class represents the main character in the game. This class

extends Entity and has methods for movement, attacking, and drawing to the screen.

**Function Specifications:**

The Player class extends the Entity class and gives it the ability to take keyboard input to move and attack. Depending on the arrow key which is pressed, the Player will move in that direction by using a set of booleans to describe the direction in which the player is moving. If the spacebar is pressed, the player will attack in the direction which it is facing. The attack is represented by a Rectangle object which is checked for intersection with all enemies on the map. Lastly, is the ‘p’ key is pressed, a boolean indicating whether the game is paused or not will be changed. The Player class also implements a series of delays and cooldowns to make sure there is cooldown between each attack and also cooldown before the player can be attacked again.

**GameState.java**

**Description:**

Abstract class for all GameStates in the game. Includes the necessary methods including initialization, update, and drawing to the screen.

**Function Specifications:**

The methods included in all classes which extend GameState are update(), draw(Graphics2D g), keyPressed(int k), keyReleased(int k), and nextState(int i).

**GameStateManager.java**

**Description:**

The class uses an ArrayList to hold all the GameStates so they can be accessed both forward and backward. Can switch from state to state easily.

**Function Specifications:**

The GameStateManager class uses an ArrayList to keep a record of all initialized game states and be able to switch from state the state easily. The setLevelState(GameState gs, int level) switches to that level if it has already been initialized or creates a new instance of it. All past levels are kept in the ArrayList because the player is still able to return to this level. The GameStateManager has update, keyPressed, keyReleased, and draw methods which calls the corresponding methods for the active game state.

**HelpState.java**

**Description:**

Extends GameState and shows the instructions on how to play.

**Function Specifications:**

The HelpState creates a background with an image containing the instructions to the game. When any key is pressed, the GameStateManager goes back to the menu screen.

**Level[X]State.java**

**Description:**

Extends LevelState and adds all enemies. Sends the player to the next level depending on what door they go through.

**Function Specifications:**

Each Level[X]State (1-8) adds the appropriate enemies to the map in the constructor. Each class also extends the nextState(int i) method to send the player to the next or previous level depending on which door was taken. If level 8, the game is moved on to the win screen.

**LevelState.java**

**Description:**

Base class for all Level[X]States and extends GameState. Has all common

functions and variables including loading in the tileMap and player and updating and drawing everything in the level.

**Function Specifications:**

The LevelState class is the base for all levels and is in charge or loading in the proper map and tileset depending on the level. The class has a variety of booleans to keep track of whether the game has been won, lost, or paused. Also, the class keeps an ArrayList of all active enemies in this specific level and is used to check collision with the player. The class calls the update and draw methods for the background, player, and all active enemies. Lastly, the LevelState draws the win or lose screen if needed and then proceeds back to the menu screen.

**MenuState.java**

**Description:**

This class represents the menu of the game and extends GameState. The menu state has 3 options which allow the users to start the game, read the help manual, or exit the game.

**Function Specifications:**

The MenuState has a background image which and draws a list of options on the screen. The options, ‘Play’, ‘Help’, and ‘Quit’, are kept in an array of Strings. The MenuState uses keyboard input to move the selection up and down and changes the game state or quits the game when the return key is pressed, depending on the option which was chosen.

**GamePanel.java**

**Description:**

This class represents the panel which holds everything in the game. Uses a

thread to update and render the game at 60 fps.

**Function Specifications:**

The GamePanel creates a JPanel and adds the necessary things for the game to function. The GamePanel has various methods to start and stop background music as well as other sound effects. The class uses a thread to update and render the screen at a rate of 60 frames per second. Each time, the draw and update methods are called for the GameStateManager.

**ProjectGon.java**

**Description:**

Main class of the entire project. Creates a JFrame and adds an instance of

GamePanel inside.

**Function Specifications:**

This class is the main file to be run when the game is played. ProjectGon creates a JFrame, positions it on screen and then adds a new GamePanel.

**JUnitTest.java**

**Description:**

A testing class for most of the methods and functions in the project.

**Function Specifications:**

This class tests the other classes by creating instances of the objects and testing to make sure each method works as it is supposed to.

**Background.java**

**Description:**

Represents a background for a gameState. Uses an image as a background and can be moved around and drawn at different positions on the jPanel.

**Function Specifications:**

This class takes in an image to use as the background and can set it to different positions on the screen. The movement in the x and y direction can also be set to create a scrolling effect.

**Tile.java**

**Description:**

Represents a single tile in the map. Each tile has a type to tell the game if

it can be walked through and if it should send the player to a new level.

**Function Specifications:**

Each tile has an image and number associated with it. The number signifies what type of tile it is: block, door, or normal. Each tile is held in the TileMap class.

**TileMap.java**

**Description:**

This class represents a map. Uses a .jpg file to load in all tile images and

then uses a .map file to read in the map. Sets the appropriate tiles to the

right places.

**Function Specifications:**

This class takes in both an image file that contains all the tiles and a text file representing the map. Using the main image, the class makes subimages for the different tiles. These tiles are then put together to create a map by reading in the map file.