Michael Chung

REDACTED Game Design Documentation

REDACTED is a role-playing game where the player explores an overworld from a top-down perspective. The player will encounter random enemies in instanced battles. Battles happen in real-time, and the player will have to use abilities bound to number keys in order to defeat enemies. These abilities will all be based on cooldown timers.

Class: GameApplication

The main application will be the stage that will transition between all game states, including a title screen, the world map, battle instance, and a game over screen. GameApplication also tracks the current game difficulty mode. This is the beginning for the application and initializes with the title screen.

Class: IntroState

This scene will set the layout for the title screen. It includes a timer that reserves interaction until the UI controls appear on the screen. This state also contains the options and the help panes. The intro state will add and remove KeyEvent handlers from the GameApplication to enable functionality of the menu selection from the title screen and the options screen.

The options screen allows for selection of game difficulty. Mode selection in the option pane works only at the launch of the game. (This can be fixed by implementing a method to reinstantiate enemies and abilities on the GameState, but could not due to time constraints.)

The help pane is a static pane that details instructions on how to play the game.

Class: GameState

This represents the scenes that include the world map. All instances of player classes and animations and enemies are created here. Key events for actions and movements will also be handled here. Movement, collision, and animation will all be handled in an animation timer. Enemies will spawn depending on which mode the game is in. GameState will transition between BackgroundLayers when the player moves to an exit point on the map.

Class: BackgroundLayer

This represents a new area on the GameState. It is responsible for spawn locations for entrances and exits. It will also keep track of the current count of enemies so that it doesn’t duplicate enemies or respawn dead enemies.

Class: BattleState

A scene that represents a battle instance. The player and the enemy that the player collides with will be represented one on one in this scene. The enemy attacks periodically while the player must use the number keys to activate abilities to bring the enemy’s health bar down to zero, at which the player will return to the GameState with the defeated enemy removed. The player may also use Run which gives a chance for the player to exit the BattleState without the enemy being removed. Upon exiting, the player will have a short time where he is invincible to entering another BattleState.

Class: DeathState

A scene representing failure for the player. This occurs after the BattleState if the enemy brings the player’s health down to zero. Pressing enter will loop the game back to the title screen, where the user can again enter GameState. Enemies defeated will not respawn and the game difficulty mode can not be changed.

Class: Entity

Most classes will inherit from this abstract class. This class should contain key information to be implemented such as images and SpriteAnimations. It ensures that entities become animated.

Class: Frog  Entity

This is the player controlled entity in the game. It also should contain an array of abilities that the player can use during battle instances. Frog should have an image view that is created from the image of the entity. Like entity, it should have a SpriteAnimation that takes its image view. Frog also has another set of images and animations that correspond to battle instances.

Class: Reaper  Entity

A basic enemy in the game. setMovement() methods ensure an AI path for the Reaper.

Class: Deceased  Entity

Similar to Reaper. Basic enemy class.

Class: SpriteAnimation

This class is used to animate players in the game. Each entity will have to have this as a private, instance variable. This takes in an ImageView that should be present in the class that it belongs to.

Class: StatusBar

This is used to visualize the player’s health, experience, and power resource. This is the base class which other classes will inherit from. It has methods to animate based on the deduction or addition of points.

Class: HealthBar  StatusBar

This inherits from StatusBar, which visualizes players’ health. The color of the bar will be set to red. They will have variables that are depended on the specific entities’ health variable.

PowerBar and ExpBar were unable to be implemented in this version of the game.

Class: PowerBar  StatusBar

This inherits from StatusBar, which visualizes players’ power resource. The color of the bar will be set to blue. They will have variables that are depended on the specific entities’ power variable.

Class: ExpBar  StatusBar

This inherits from StatusBar, which visualizes players’ experience to the next level. The color of the bar will be set to yellow. They will have variables that are depended on the specific entities’ current level.

Class: Ability

Ability is an abstract class that is represented on screen as icons in the BattleState. They will use key events (number keys) to activate. They should all have global cooldown times that will activate once any ability is pressed. Each ability also has a local cooldown that is longer and will activate when the ability is used. Each will have a different effect that should be determined in their sub classes.

Class: BasicAttack  Ability

Inheriting from Ability, this does 10 damage to the enemy.

Class: AdvancedAttack Ability

This ability does 20 damage to the enemy.

Class: FireDamage  Ability

This ability does 15 damage to the enemy. (Unimplemented is bleed damage that does damage over time for a limited time)

Class: BasicHeal  Ability

This ability restores 10 health to the player.

Class: Run  Ability

This ability gives a chance for the player to exit the battle instance based on random number generation. The specifics of escaping are detailed in the BattleState class description.

All necessary files for the game to run properly, including music and images, should be found in the resources folder.