CSE443 – OBJECT ORIENTED ANALYSIS AND DESIGN

HOMEWORK 1

REPORT

My Design Decision Explanations:

- Background

I used some math for the background continuity. I draw 2 background image on the game window according to player's coordinates. When the player is halfway through the first image, I need to draw second image. If I don't do this, the player will se a empty background. In the same way, when the player is halfway through the second image, I need to draw first image again. I was able to provide this continuity by taking mode (%) according to the length of the 2 images, when the player is halfway through the images.

- The character actions

I implemented the KeyListener. So I listen the key inputs of the player. If the player presses the D key, the background and immobile entities move to the left. Also if the player presses the space key, the player jumps (initially low jump).

- Game window debug/text messages part

I draw the background image on the part of the game window. I draw messages (String) fort he remaining part. When the message window is full, I clean it and draw again from the beginning.

- Types of jumps – Strategy design pattern

I create an JumpBehavior interface. The player has a JumpBehavior. The LowJump and HighJump classes implement the interface. The jump of the player is determined at the run time. When the type D power up is acquired, the jump mode changes.

- Lose a life

If the player collide a Monster, the player lose a life. One collision only reduces the player's life once. I did this with a flag (collisionMonster). If there is a collision, the flag becomes true. The flag remains true until the player passes the Monster. If the player passes the Monster, flag becomes false and this controlling can happen again.

- Successful jump – earn points

If the next Monster is now previous Monster, I check if there is a collision. If the player collided with the previous Monster, the player does not earn the points. If the player did not collide with the previous Monster, the player earns the points. The player earns the points as current power factor (points += score multiplier). The power factor increases,

when the player acquire a power up. I'm doing it true to never check the collision variable of the monster I'm controlling again. Because if the player collides with the previous monster, the player will not get points. I'm assuming that the player collides with the monster after earning points. So the player doesn't earn points from the same monster again.

- Powerups – Decorator design pattern

I create an abstract Powerup class and an abstract PowerupDecorator class. These A, B, and C types extends the decorator class. I create a inital power up class because the decorators have not a power up that they use as initiliaze power up. They have a score mutiplier. I put the x1 score multiplier into the InitializePowerup class. So the D type class extends the powerup class because it has some behavior but it does not use score multiplier function. I add an powerups list to the Player class. I put initialize power up as first element to the list. When the player acquire a power up, I add the new power up into the list. So the power factor of the player calculate easily. The last added power up is getted and its score multiplier function calls.

- I implement the MouseListener. I used it to listen to the buttons in the menu and the buttons in the game.

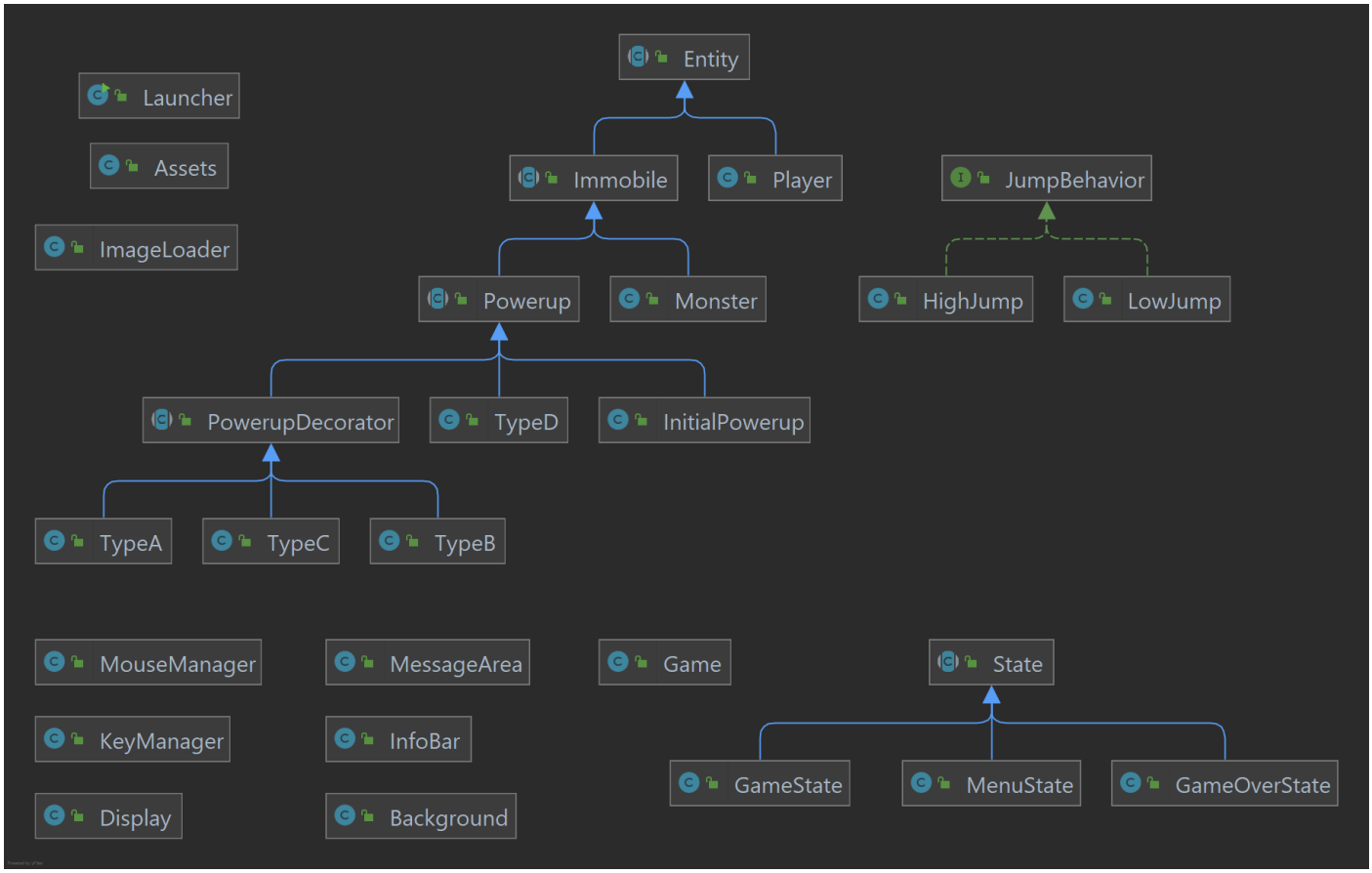
- I add four monsters and four powerups initially into the game. When an enitity leaves the game window, I removed it and added new one.

- I marked the res directory as the resource. The res directory contains my game images. I uploaded the images in the res file with getResource in the ImageLoader class.

- The Launcher class is my main class. It is launch the game.

Class Diagram:

This diagram contains only inheritance.



Şeyda Özer

171044023