Gebze Technical University

Computer Engineering Department

CSE 443 – Object Oriented Analysis and Design

Fall 2021 – 2022

Homework 1

HOMEWORK REPORT

Türker Tercan

171044032

turker.tercan2017@gtu.edu.tr

Objective

Making a 2D-Side scroller game like Mario, Google Dinosaur game with Strategy and Decorator design patterns.

There is only one main character and objective to get the highest score by jumping over obstacles or monsters. Character has three health and certain amount of speed. It has two jumping techniques which are short and long jump. Implement these two techniques using strategy pattern.

There are four types of power-ups that the main character can obtain if it collides with them.

- Type A: x2 score boost
- Type B: x5 score boost
- Type C: x10 score boost
- Type D: Jump behavior changer

Type A, B, and C power-ups are cumulative, so if you have Type A after Type C, the character's score multiplier will be x20. Use decorator design pattern to implement this feature so if we want to add a new power-up to the game it will cost a lot less.

Graphics and Design

I used Super Mario Worlds' classic background. For the characters and enemies, I borrowed the sprites from Undertale. I drew UI buttons and power-up sprites.

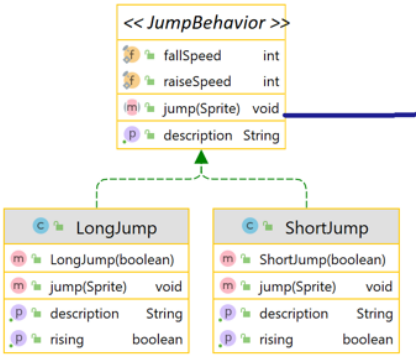
Implementation

I used Java Swing library to make functionalities of the game. I implemented my own panel which extends from JPanel and keeping the tracks of mouse and keyboard, I've implemented KeyboardListener and MouseListener interfaces for GamePanel class. Also, it implements Runnable. I used GamePanel as a thread to fix the fps at 60. Its re-renders and makes game logics in every frame.

Since I defined my GamePanel is a Runnable class. I initialized a thread to run update and render methods at fixed fps. Thus, if our game logic somehow stocks during gameplay, other thread would be able to listen keyboard and mouse.

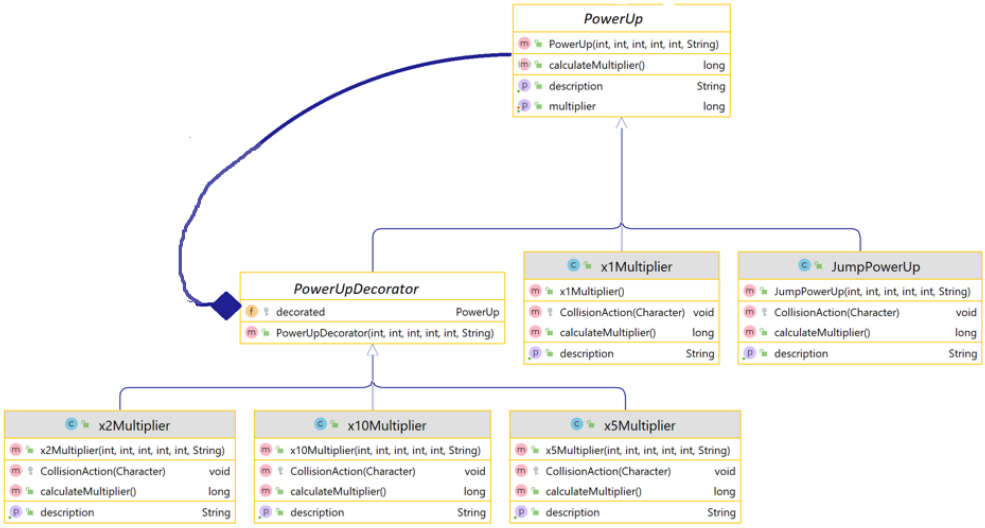
Strategy Design Pattern

JumpBehavior interface defines jump method and LongJump and ShortJump classes extends from JumpBehavior class. LongJump and ShortJump makes the main character jump. Since they implemented with strategy pattern it can be switch to in runtime.



Decorator Design Pattern

PowerUp is an abstract class, and it has a long variable to store its multiplier value. X1Multiplier and JumpPowerUp classes are concrete classes which are extended from PowerUp class.



PowerUpDecorator is also abstract class, and it extends from PowerUp. The difference is it has also a PowerUp component to know which PowerUp is decorated. CalculateMultiplier method is multiplies current object's multiplier with decorated object's multiplier like recursively to get the main characters multiplier.

My Sprite class is an abstract class and it just helps to display the object's image and keep the track of the image's coordinates. Also, it implements moveLeft method to move the object to the left.

Collidable class extends from Sprite class and it is also an abstract class. It defines an abstract method which is called CollisionAction. When an object like Enemy, PowerUp collided with another object this method will be used.

For the further information about implementation and classes, there are UML Class Diagram and JavaDoc in this current folder.