CSE 443 – Object Oriented Analysis and Design

Homework 1 Report

İlkan Mert Okul 1801042649

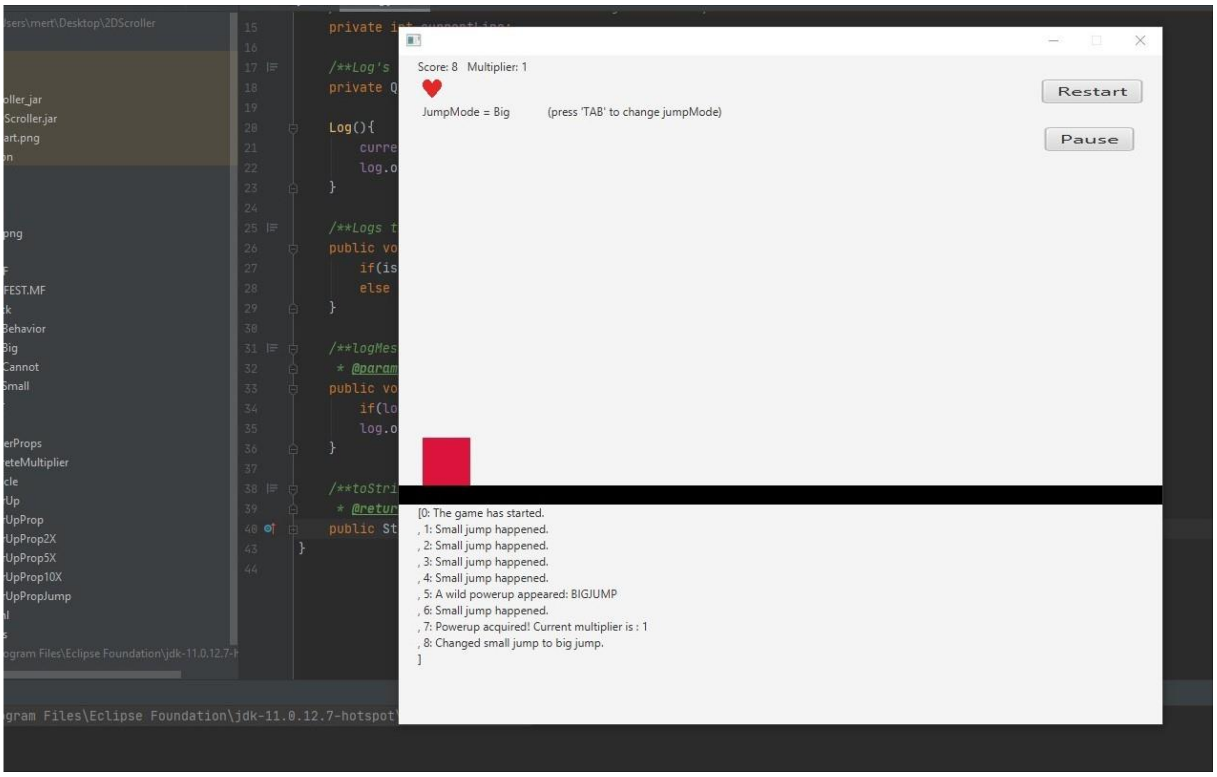
How to play the game?

The game works a little bit different. Since i could not be able to process multiple key inputs, ‘D’ key spawns new obstacles and powerups and they are automaticly moves to the left.

So you need to TAP to ‘D’ key, NOT press and hold.

Jumping mode is more functional. After player take powerup for Big Jump, can press ‘TAB’ to swap jump types.

‘TAB’ key can help player to swap between jumps.



```
[0: The game has started.
, 1: Small jump happened.
, 2: Small jump happened.
, 3: Small jump happened.
, 4: Small jump happened.
, 5: A wild powerup appeared: BIGJUMP
, 6: Small jump happened.
, 7: Powerup acquired! Current multiplier is : 1
, 8: Changed small jump to big jump.
]
```

You can find every single method’s explanation in source code or javadoc.

Strategy Pattern:

Class ‘Player’ is the character that user plays. That class can Jump. And it is done by ***Strategy pattern.***

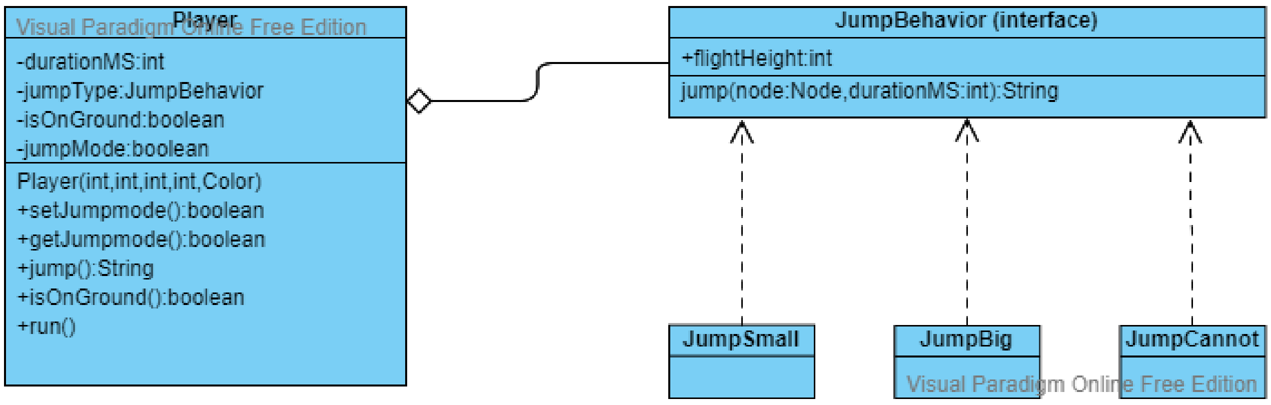
We have three jump types**: JumpBig, JumpSmall, JumpCannot.**

Those jump types are concrete classes that implements JumpBehavior interface, which has jump method’s declaration.

So ‘Player’ has a JumpBehavior reference that changes back and forth to act differently in different situations.

As said, when Jump powerup acquired, player can change its behavior to JumpBig behavior that makes the player jump higher and stay in air longer.

When the player is in air, player cannot jump again, so player enters to JumpCannot behavior.



Those class diagrams are avaiable in ClassDiagram folder too.

Decorator Pattern

Powerup picking is done by Decorator pattern, because multipliers adds up all the time. And we might need to add new powerup types to add on. This pattern helps.

So PowerUpProp is the base abstract class, and every concrete PowerUp prop extends it.

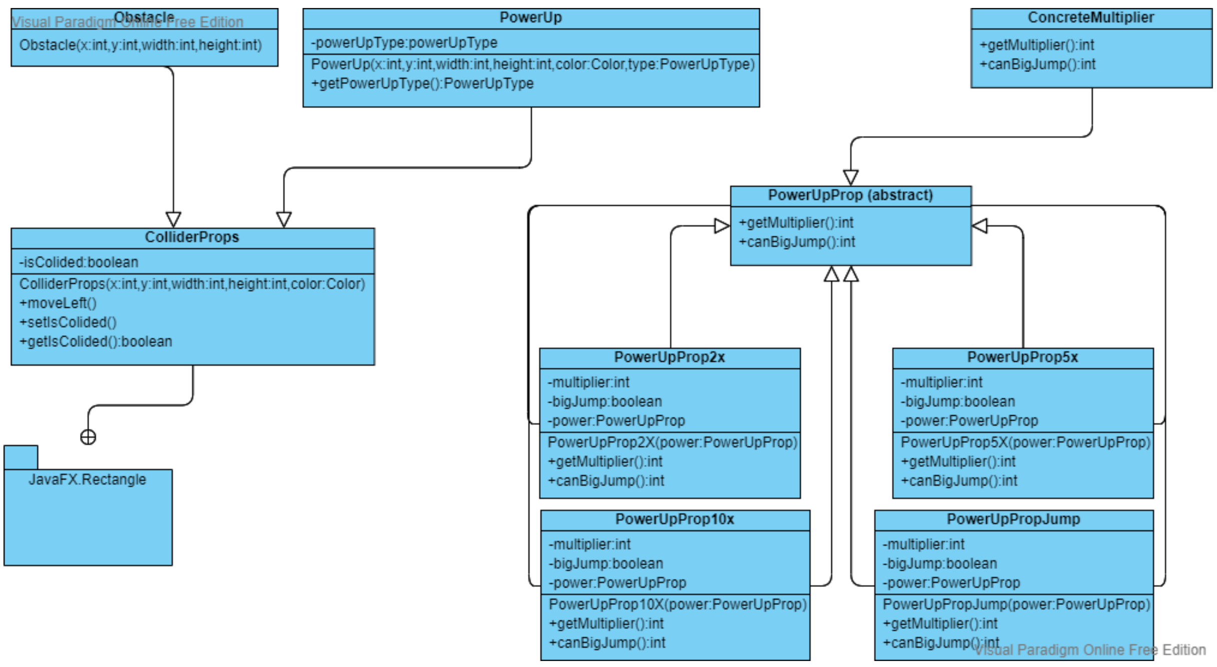
Every concrete PowerUp prop is and has a base PowerUpProp. This way we can add up old props and 'decorate' our multiplier(int) and jump(boolean) variables.

This way we can know the current multiplier by :

return this.multiplier * power.getMultiplier();

where power is the old PowerUpProp that got decorated.

Right now, we have 4 types of PowerUpProps : 2x, 5x, 10x, Jump and all of them is in this pattern. And we can add any powers we want anytime.



Worth to Mention

The difference between PowerUp and PowerUpProp is, PowerUp is an object that we see on screen, and it has PowerUpProp component to know what multiplier it holds in itself.

ColliderProps class is a base class for both PowerUp (not PowerUpProp) and Obstacle.

They are both collideable with player so they have a common, and they are both rectangle.

Obstacle is created with random color every time. PowerUp is always has the color of cyan.

<u>Thanks for reading, İlkan Mert Okul, 1801042649</u>