CSE 443
Object Oriented Analysis and Design

Homework - 1

Muharrem Ozan Yeşiller
171044033

Project Definition

Project is a 2D Side scrolling game. While our game character is trying to escape from an area, some monsters and power-ups appear in front of him. If the character is caught by monsters, he loses one of his 3 lives. Increases the score point coefficient if the character gets buffs. If the character picks up the shoe he finds on the road, he will jump higher. Important note, if the character takes the shoe for the second time, this time he will return to his old jumping power.

How to Play?

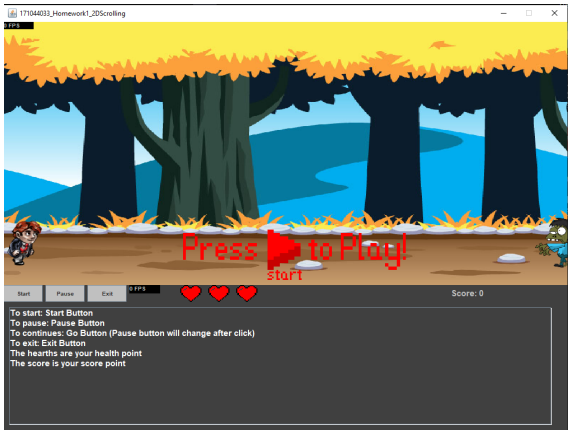
First of all, the start button is pressed and the game starts. The character runs by pressing the 'd' key and jumps by pressing the 'space' key. The user can pause the game with the 'pause button'. Pause button turns into 'Go Button' When the Go button is pressed, the game continues. The user can close the game with the exit button.

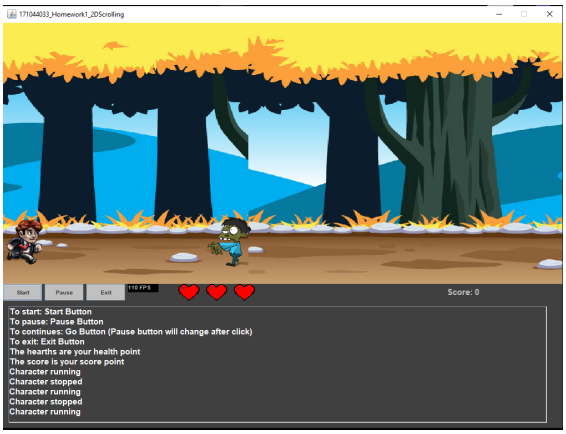
When the user loses all their lives, they can start the game by saying start again.

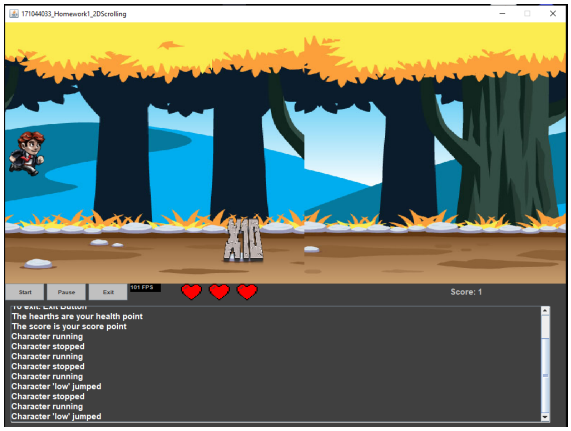
The user can see the FPS bar in the upper left. But the background image advances at certain intervals and there is a fps bar next to the buttons to make it clearer.

Non Functional Requirements

Java Development Kit 14 (JDK 14)
An average computer and operating systems (Windows/Unix/macOS)
Keyboard, ‘d’ and ‘space’ should work. (if default)


171044033_Homework1_2DScrolling
Press to Play!
start
Start
Pause
Exit
Score: 0
To start: Start Button
To pause: Pause Button
To continues: Go Button (Pause button will change after click)
To exit: Exit Button
The hearths are your health point
The score is your score point


171044033_Homework1_2DScrolling
Start
Pause
Exit
Score: 0
To start: Start Button
To pause: Pause Button
To continues: Go Button (Pause button will change after click)
To exit: Exit Button
The hearths are your health point
The score is your score point
Character running
Character stopped
Character running
Character stopped
Character running


171044033_Homework1_2DScrolling
X10
Start
Pause
Exit
Score: 1
The hearths are your health point
The score is your score point
Character running
Character stopped
Character running
Character stopped
Character running
Character 'low' jumped
Character stopped
Character running
Character 'low' jumped


171044033_Homework1_2DScrolling
Pause
Start
Go
Exit
Score: 1
Character running
Character 'low' jumped
Character stopped
Character running
Character 'low' jumped
Character stopped
Character running
Character get 'X10 BUF'
Character 'low' jumped
Game paused
Character will stop

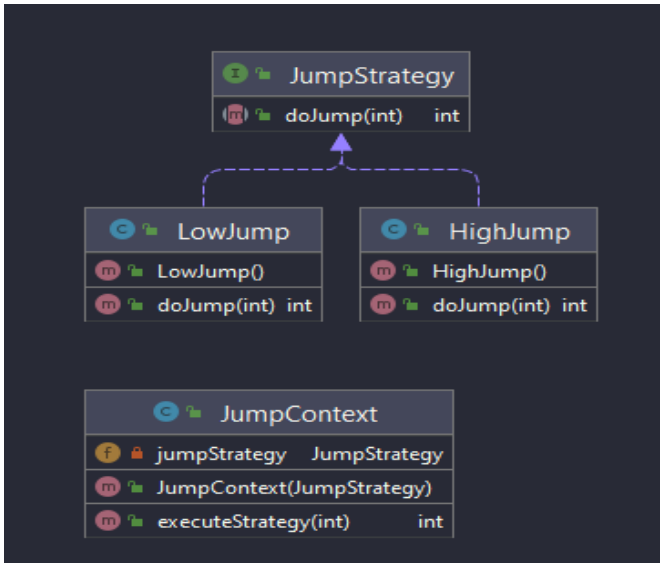

171044033_Homework1_2DScrolling
Start
Pause
Exit
102 FPS
Score: 501
Character 'low' jumped
Character 'low' jumped
Character 'low' jumped
Character 'low' jumped
Character 'low' jumped
Character 'low' jumped
Character 'low' jumped
Character 'low' jumped
Character lost 1 health
Character 'low' jumped
Character 'low' jumped

Design Patterns and Class Diagrams

Strategy Design Pattern

In our software, more than one algorithm may exist to perform an operation. The strategy design pattern is used to select and implement a method based on the situation. Each algorithm is implemented for a class. So to sum things up, the behavior or algorithm of a class at runtime can be changed according to a strategy.

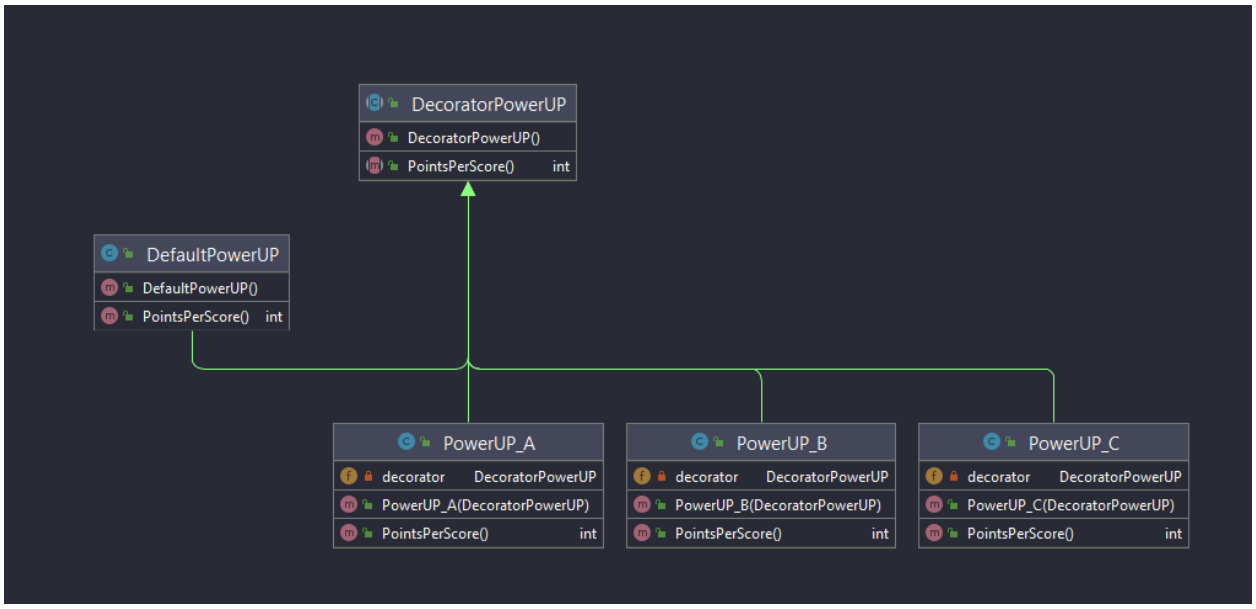
In order to implement the strategy design pattern, we need to define an interface named Strategy. This interface contains a method or methods to be implemented by subclasses. Our strategy here is to jump high or low.




```
if (highJumpEnabled) {
    main_char.setJumpContext(new JumpContext(new LowJump()));
    logArea.setText(logArea.getText() + "\nCharacter get 'low jump'");
    highJumpEnabled = false;
} else {
    main_char.setJumpContext(new JumpContext(new HighJump()));
    logArea.setText(logArea.getText() + "\nCharacter get 'high jump'");
    highJumpEnabled = true;
}
```

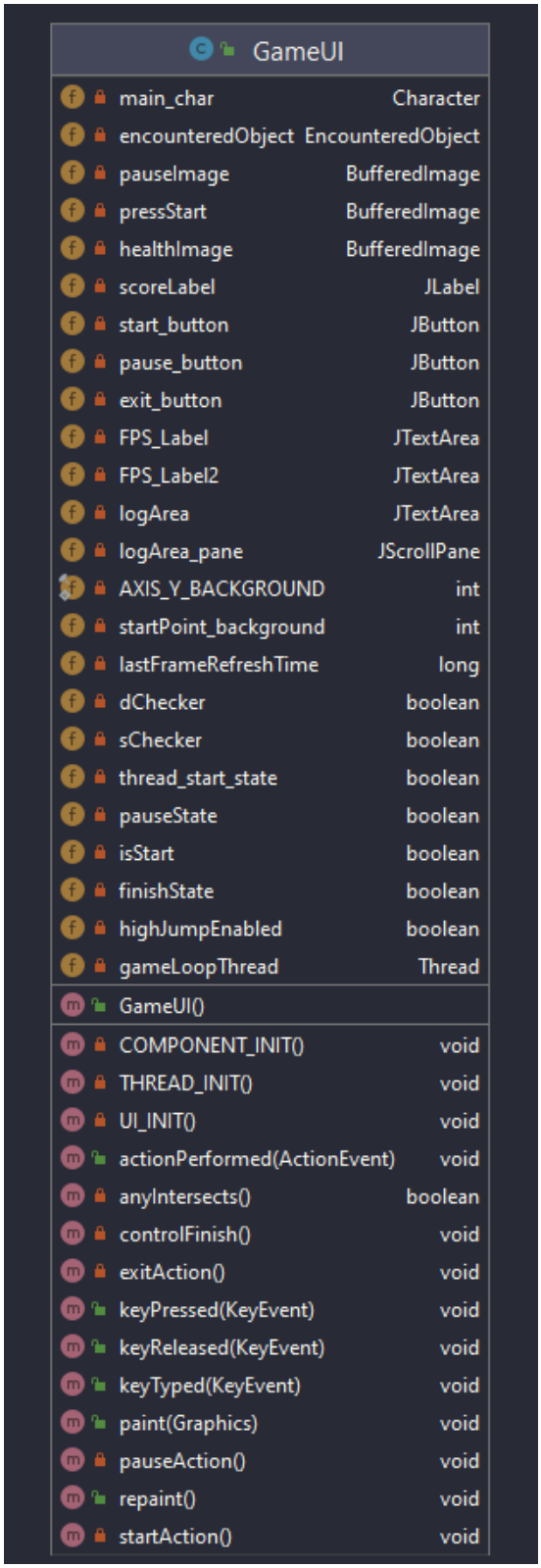
Decorator Design Pattern

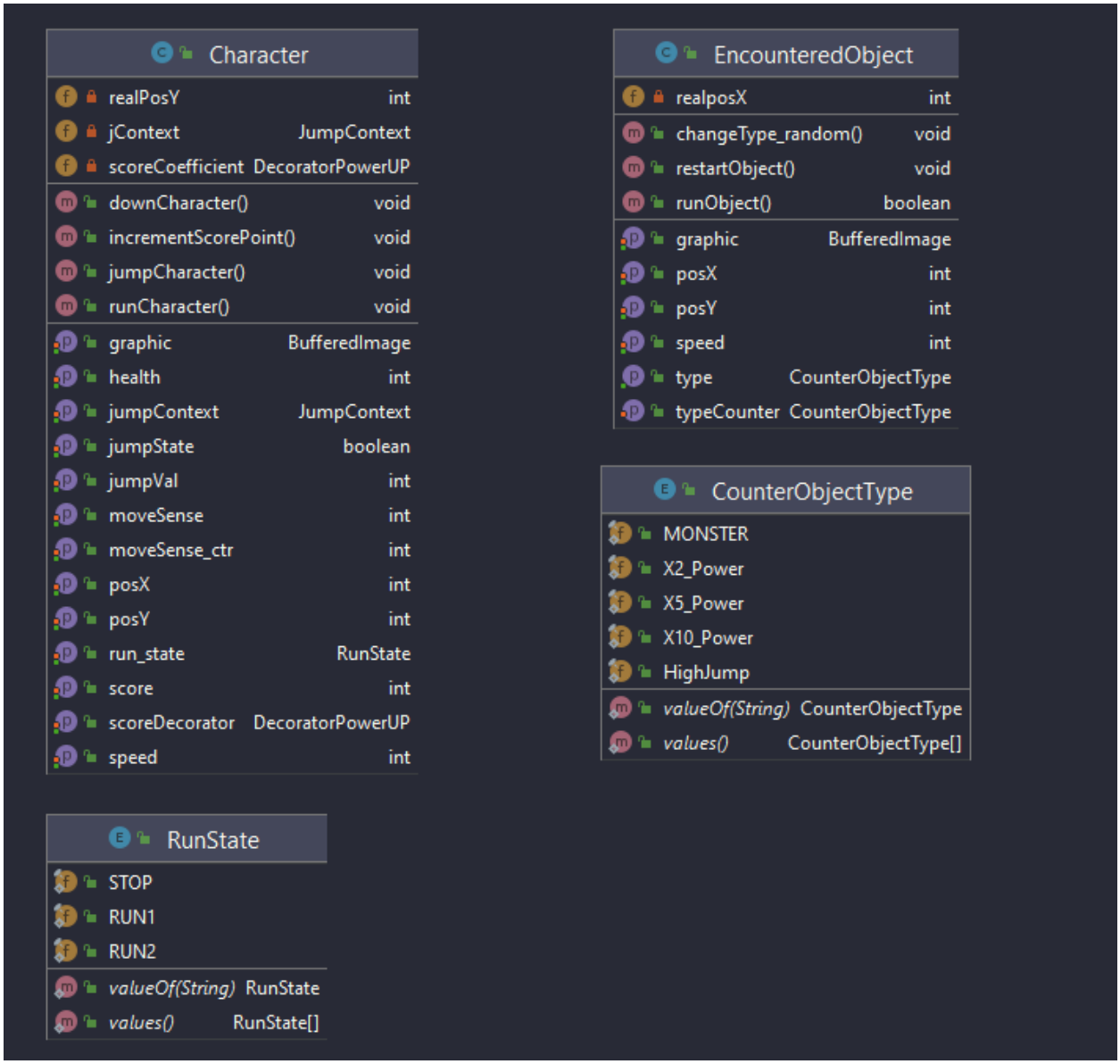
When we want to add extra features without modifying our objects, it is the pattern that we can use these features by wrapping them in decorator objects. It is especially used when different decorations are needed for an object. Although it is simple to use, the trick is to encapsulate an object that implements the same protocol.




```
} else if (encounteredObject.getType() == CounterObjectType.X2_Power) {
    main_char.setScoreDecorator(new PowerUP_A(main_char.getScoreDecorator()));
    logArea.setText(logArea.getText() + "\nCharacter get 'X2 BUF'");
} else if (encounteredObject.getType() == CounterObjectType.X5_Power) {
    main_char.setScoreDecorator(new PowerUP_B(main_char.getScoreDecorator()));
    logArea.setText(logArea.getText() + "\nCharacter get 'X5 BUF'");
} else if (encounteredObject.getType() == CounterObjectType.X10_Power) {
    main_char.setScoreDecorator(new PowerUP_C(main_char.getScoreDecorator()));
    logArea.setText(logArea.getText() + "\nCharacter get 'X10 BUF'");
}
```

Other Classes




Character
realPosY int
jContext JumpContext
scoreCoefficient DecoratorPowerUP
downCharacter() void
incrementScorePoint() void
jumpCharacter() void
runCharacter() void
graphic BufferedImage
health int
jumpContext JumpContext
jumpState boolean
jumpVal int
moveSense int
moveSense_ctr int
posX int
posY int
run_state RunState
score int
scoreDecorator DecoratorPowerUP
speed int
EncounteredObject
realposX int
changeType_random() void
restartObject() void
runObject() boolean
graphic BufferedImage
posX int
posY int
speed int
type CounterObjectType
typeCounter CounterObjectType
CounterObjectType
MONSTER
X2_Power
X5_Power
X10_Power
HighJump
valueOf(String) CounterObjectType
values() CounterObjectType[]
RunState
STOP
RUN1
RUN2
valueOf(String) RunState
values() RunState[]