Gebze Technical University
Tugay Apaydın
1801042081

CSE443 - OOAD
Homework 2 - 2021

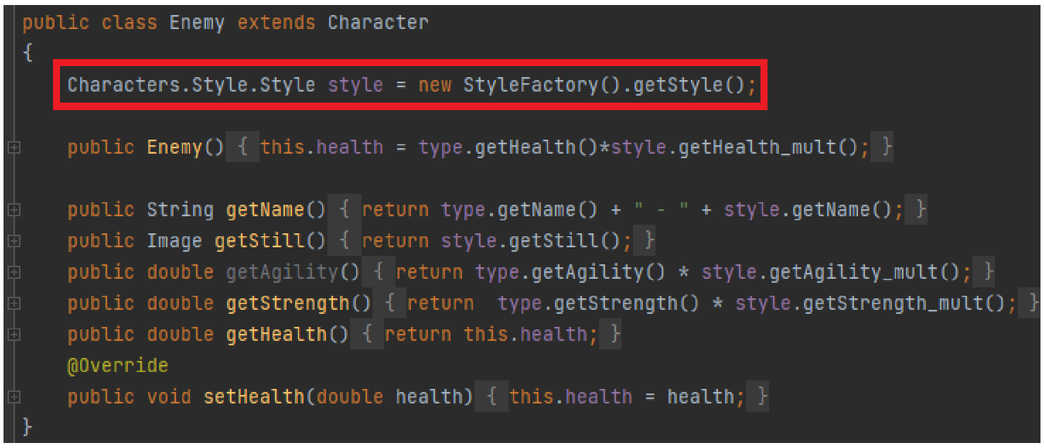
Report

Key Functionalities: ESC key pauses the game and prints main menu at the screen. The resume button is only available after the game is started.

Abstract Factory Design Pattern

In the project, there are 2 types of factories: TypeFactory and StyleFactory. These factories implement AbstractFactory interface and are responsible to create Type and Style abstract class objects. The base types that are mentioned in the pdf (Fire, Naure, Ice) extend Type abstract class and the styles (Valhalla, Underwild, Atlantis) extend Style abstract class.

There are 2 types of characters: Enemy and Char. While char can only have Type class object, Enemy can have Type and Style class object.



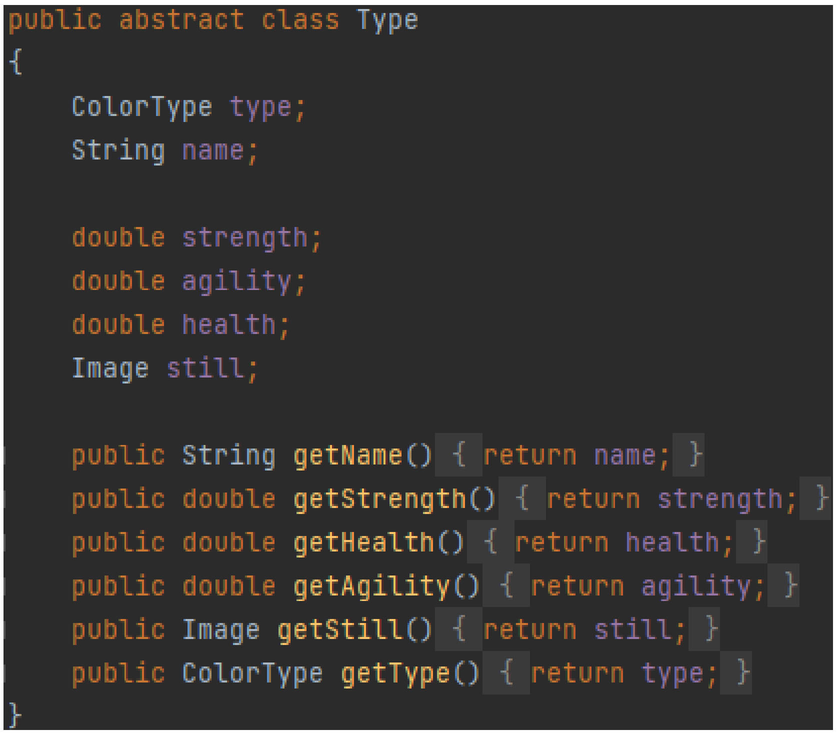
```java
public class Enemy extends Character
{
    Characters.Style.Style style = new StyleFactory().getStyle();

    public Enemy() { this.health = type.getHealth()*style.getHealth_mult(); }

    public String getName() { return type.getName() + " - " + style.getName(); }
    public Image getStill() { return style.getStill(); }
    public double getAgility() { return type.getAgility() * style.getAgility_mult(); }
    public double getStrength() { return  type.getStrength() * style.getStrength_mult(); }
    public double getHealth() { return this.health; }
    @Override
    public void setHealth(double health) { this.health = health; }
}
```

Figura 1: Style Creation with StyleFactory

In figure 1, you can see the usage of the StyleFactory. StyleFactory and TypeFactory have the similar implementations. Also you can see that, in order to get the properties of a character (strength, health, agility, image of the character etc.), Style and Type objects are used.



```java
public abstract class Type
{
    ColorType type;
    String name;

    double strength;
    double agility;
    double health;
    Image still;

    public String getName() { return name; }
    public double getStrength() { return strength; }
    public double getHealth() { return health; }
    public double getAgility() { return agility; }
    public Image getStill() { return still; }
    public ColorType getType() { return type; }
}
```

Figura 2: Abstract Type Class

The types (Fire, Ice, Nature) extends the abstract Type class shown in Figure 2. Their properties are defined in class attributes.

Gem Creator Class

The GemCreator class is responsible to handle game panel. It's main function updateList() is called at each frame and it handles the operations and their animations: shifting gems, removing gems, shifting gems upwards after removing, computer's turn etc. The game starts after the first shifting and the GemCreator class starts handling the operations. After that all operations are handled in a sequence: shuffling, checking for matched tiles, filling empty spots after removing matched tiles etc.

Important Bugs and Missing Parts

If a match occurs and some tiles are removed the upshifting operation (which shifts gems up to fill empty spots) is handled. But after that, in order to fill empty tiles at below, the fillEmptySpots function is called. The problem is, this fillEmptySpots function just randomly fills empty spots. It does not check if the new randomly created tiles causes other matches.

Although this function includes this type of a bug, in the shuffle function (which creates a new game table) handles the situation well and does not create matching tiles at start.

Another issue is, the damages are abnormal. Characters may die with just a single damage. I suspect that there is some wrong coding in this part.

Other missing part is, when characters are all dead, the game will not restart. It must be restarted again from the main menu (ESC key opens the menu).

Character Panel

The character panel only includes character definitions and damage functionalities. The damage function parses the removed tiles and determines which character/enemy to be damaged.