Java-Project Documentation

Content:

1. Design
2. Use cases
3. Design decisions
4. Testing
5. References
6. **Design**

The favorite game I played in my childhood is “Raiden”, which describe battles of warplanes. It is a popular game at that time. I spent lots of holiday time on it and it gave me unforgettable happy memory. In this project, I have opportunity using JAVA to build my own “Raiden “game.

Game interface:

Hero’s plane, Enemy planes, Bullet from each plane, Background, Score.

Design flow:

1. Design game window.
2. Add game background.
3. Realize background animation to show flight effect.
4. Add hero plane to the window.
5. Using keyboard to control hero plane.
6. Add enemy plane and fire automatically.
7. Collision detection of bullets and aircraft.
8. Realize hitting the plane to produce an explosion effect.
9. Realize hitting the enemy plane and disappeared.
10. Add enemy automatically.
11. Add background, bullet, and boom music.
12. Pack the game as an executable program.
    1. **Language Capabilities**

Abstract class:

电脑萤幕的截图

描述已自动生成

Enums:

电脑萤幕的截图

描述已自动生成

Inheritance:

电脑萤幕的截图

描述已自动生成

* 1. **Design Patterns**

This program applied **Factory Method Design Pattern**.

The enemy and our plane have similarities and differences. If the enemy plane class is created on the basis of the Plane where I put our plane, it is necessary to modify the existing plane class.

Object-Oriented Programming Should Follow the "Open-Closed Principle".

Open-Closed Principle: Software entities should be open to extension of functions and closed to modification.

When extending functions, we should extend the behavior of our software entities, not modify existing code, otherwise it may affect or destroy existing functions.

UML shows below:

图形用户界面, 文本

描述已自动生成

Make enemy plane and our plane have a same parent class – plane class.

Factory Method Design Pattern implements by using below association:

|  |  |
| --- | --- |
| Product | Plane |
| Concrete Product | OurPlane, EnemyPlane |
| Factory | PlaneFactory (interface) |
| Concrete Factory | OurPlaneFactory, EnemyPlaneFactory |

It is very convenient to directly expand new models without modifying the original code and facilitate subsequent maintenance work.

1. **Use cases**

Actors: Player, System

At leisure time, a player opens Warplane game to continue to hit the highest score. Player clicks java program and chooses start game. He manipulates the hero plane to fight with enemy fighters. Every time the player hit in a target; he will get 100 credits.

Basic flow:

* + 1. Player manipulates plane to move, use up, down, left, and right directions by typing keyboard. Press space key to fire the bullet hit enemy fighters.
    2. Player controls the plane to prevent being hit by randomly appeared enemy fighters.
    3. Player gets 100 credits by hit enemy fighters.
    4. Player gets hit by enemy bullet then game over.

图示

描述已自动生成

1. **Design decisions**

**Plane Wars Class**

Plane Wars extends JFrame, using window function JFrame to create game windows. Applied init() method to manage all the functions, do the initialization. AddBackground method to create the game background, set the background picture. Control method is used to control the movement of the plane, up, down, left and right by hitting the keyboard direction keys. Besides, space in the keyboard was used to fire bullet. Enemy planes are more than one, so I create a ArrayList to contain enemy planes. Crash and crashHero are used to detect collision. Use the rectangle of the object, it depends on the picture that we set. When the bullet rectangle and plane are intersected, it means plane was hit and execute the corresponding action. For the sound method, I use this method to set the source of the music, when we need sound, just call this method by using string argument.

电脑萤幕的截图

描述已自动生成

**PlaneFactory interface**

At first, I did not apply interface, just let hero plane and enemy plane to inherent the plane class. The enemy and our plane have similarities and differences. If the enemy plane class is created on the basis of the Plane where I put our plane, it is necessary to modify the existing plane class.

So I decide to use design pattern, **Factory Method Design Pattern.** This interface has two methods, createPlane with argument and createPlane without argument.

电脑萤幕的截图

描述已自动生成

**OurPlaneFactory and EnemyPlaneFactory classes**

These two classes implement PlaneFactory interface. It specifically creates plane methodologies. The difference is location of creation, enemy planes’ locations are random, but hero plane is not. So enemy plane needs argument method.

电脑萤幕的截图

描述已自动生成

电脑萤幕的截图

描述已自动生成

**Plane class**

This is an abstract class which is the father of hero plane and enemy plane. In this class, It includes the method for get and set location information. SetPlaneImagePath method, drawPlane mehod, getPlaneRectangle method and explode method are including. Then, fire and getBulletRectangle methods are different for hero plane and enemy plane, so they are abstract method which waits the class to specifically implies.

电脑萤幕的截图

描述已自动生成

**OurPlane and EnemyPlane class**

These two classes extend plane class. They are the son class of plane class. They need to create fire and getBulletRectangle methods. For hero plane, fire bullet is controlled by player, but enemy plane is automatically fire. For bullets, hero plane has double bullet when fire, but enemy only have one. So they need different method to create.

电脑萤幕的截图

描述已自动生成

电脑萤幕的截图

描述已自动生成

1. **Testing**

In testing part, I applied **Junit5** to write unit test.

Unit testing is writing test code that uses specific, unambiguous, fine-grained functionality.

Unit testing tasks include:

**1. Interface function test: used to ensure the correctness of the interface function.**

电脑萤幕的截图

描述已自动生成

**2. Boundary Condition Test**

电脑萤幕的截图

描述已自动生成

**3. Running test**

电脑萤幕的截图

描述已自动生成

We can see from the following screenshot. Background, planes, bullets are set as we expected. Use keyboard to control the hero plane to move and fire bullet.

地图

描述已自动生成

电脑萤幕的截图

描述已自动生成

And there is no exception or error exists. It means our small game is successful.

1. **References**
2. <https://www.tutorialspoint.com/design_pattern/factory_pattern.htm>
3. https://docs.oracle.com/javase/7/docs/api/java/io/package-summary.html
4. <https://www.bilibili.com/video/BV18b4y1i7oa?p=6&spm_id_from=pageDriver>
5. <https://blog.csdn.net/kai_wei_zhang/article/details/8120382>
6. <https://www.runoob.com/java/java-tutorial.html>