FOOP Final Project Report

Members

1. 林立珩 B06902043
2. 陳暘 B06902044
3. 林采鋒 B0690261

Overview

In this final project, we create a simple warplane shooting game. The goal for the player is to beat the boss by controling a warplane. During the game, there will be some boosts falling down, by eating the boost, the warplane can increase some of its abilities (e.g. moving velocity) for a short period.

How to play

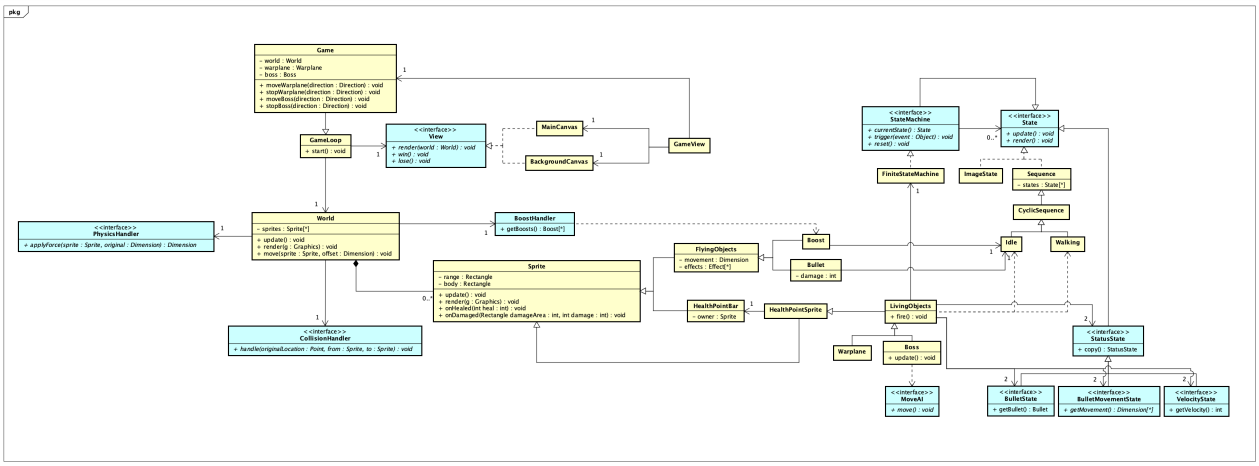
The player use W/ S/ A/ D to move the warplane up/ down/ left/ right. The bullets are fired automatically, so the player just needs to focus on moving the warplane.

Responsibility

1. 林立珩: Design/ Implementation of Sprites
2. 陳暘: Design/ Implementation of View, Controller and PhysicsHandler
3. 林采鋒: Design/ Implementation of CollisionHandler

- All members have equal contribution.

Class Diagram:



Description:

The game is based on https://github.com/Johnny850807/Java-Game-Programming-with-FSM-and-MVC, so we only elaborate the modifications/ extensions we made.

Model

1. Sprites:
    1. FlyingObject: `FlyingObject` is extended from `Sprite`. **movement** indicates the direction it moves toward, and **effects** indicates what will happend when this object is collided with a `LivingObject`. Since they will only keep moving, we only need one state `Idle` to display it.
        1. Boost: `Boost` is extended from `FlyingObject`, and they will only move downward. `World` will call `BoostHandler`'s **getBoost()** method in every **update()** call to know whether there are boosts created in this time step.
        2. Bullet: `Bullet` is extended from `FlyingObject`. `Bullet` objects will have an additional variable **damage** because they have intrinsic damage effect.
    2. LivingObjects: `LivingObject` is extended from `HealthPointSprite` with an additional abstract method **fire()**. To handle that the status is changed by boosts, every `LivingObject` will have a default `StatusState` and a current one on Cool Down (i.e. CD)/ Bullet/ Bullet Movement (i.e. how bullets move)/ Velocity (i.e. how fast the `LivingObject` moves).
        1. Warplane
        2. Boss: `Boss` is extended from `LivingObject` because boss will turn into the angry mode for a period if its HP is too low. Therefore, we should override the **update()** method. Also, since the boss should move automatically, we use `MoveAI` to control this.
2. PhysicsHandler: `PhysicsHandler` is an interface with `applyForce(Sprite sprite, Dimension offset)` method. It will update the poistion of incoming `Sprite` according to the `Dimension`.
    1. NormalPhysicsHandler:
        1. `NormalPhysicsHandler` is extended from `PhysicsHandler`.
        2. Update the position normally.
    2. RandomPhysicsHandler:
        1. `RandomPhysicsHandler` is extended from `PhysicsHandler`.

    2. Update the position randomly.
3. CollisionHandler:
    1. handle() This interface handles the collision of bodies of two instance. The handler will apply the effect of `FlyingObject` to `LivingObject`. Also, it will handle the adjustment of position offset, and remove the collisioned `FlyingObject`.

- ## View

1. View:
    1. An interface own the method **renderWorld(world)**.
2. Background:
    1. Own a list of `Image` as background image.
3. Canvas: `Canvas` is extended from `JPanel` with **render()** method to render the property `Background`.
    1. MainCanvas:
        1. `MainCanvas` is extended from `Canvas` and implements `View` interface.
        2. Render the background images of the main game view.
    2. BackgroundCanvas:
        1. `BackgroundCanvas` is extended from `Canvas`.
        2. Render the background of the start and end view.
4. GameView:
    1. `GameView` is extended from `JFrame`
    2. Communicate with `Game` controller to choose which `Canvas` to display.

- ## Controller

1. GameLoop:
    1. Control the game flow
        1. Start
        2. Win
        3. Lose
    2. Do the following steps in each loop.
        1. Update sprites
        2. Render sprites
2. Game:
    1. `Game` is extened from `GameLoop`
    2. Add addition methods to move and stop the `Warplane` and `Boss`

Advantages

1. OCP:
    1. MoveAI
    2. StatusState/ BulletState/ BulletMovementState/ VelocityState

3. BoostHandler
4. PhysicsHandler
5. CollisionHandler
6. Canvas

Disadvantages

1. We can only have predefined statuses to be changed.

References

https://github.com/Johnny850807/Java-Game-Programming-with-FSM-and-MVC.