CPS 108 VOOGA Individual Design

Name: Donghe Zhao
Date: April 20th
Genre: Fighting Game
Teammate: Hareesh, Helena, Chen, Peggy, Wendy, Hui
UTA: Nathan, Tanner
GitLink: https://github.com/hareeshganesan/Vooga/tree/master/src/PhysicsEngine

Introduction

The Physics Engine is made up by two aspects, one is physics calculation, and the other is Collision Engine. Collision Engine is the most important part in this job. I made the collision part very easy for developers to set.

In every game, we just need to initially set the collision with the following example.

1. Create a instance viable to keep my Collision

```
// Collision
private Collision myCollision;
private SpriteGroupTemplate groupSprite;
```

2. Initial set the collision. As a developer, we just need to pass in all the sprites as a group in to the myCollision. This group is SpriteGroupTemplate, which has been shown above, made up of SpriteTemplate, since I want to make it general and flexible, so I make the group based on SpriteTemplate. The other thing we need to pass in to the myCollision is the Collision kind. We can pass in a list of Collision kind or pass in CollisionKind one by one. What's more, each CollisionKind has a

reaction or some reactions associated with itself.

```
SpriteGroupTemplate groupSprite = new SpriteGroupTemplate("team");
groupSprite.addFighterSpriteArray(playerSprites);
groupSprite.addPlatformBlockArray(platform);

ArrayList<CollisionKind> CollisionkindList = new ArrayList<CollisionKind>();
CollisionkindList.add(new CollisionKindFriends(new ReactionMomentumConservation()));
CollisionkindList.add(new CollisionKindEnemy(new ReactionPush()));
CollisionkindList.add(new CollisionKindNeutral(new ReactionRebound()));

myCollision = new Collision(groupSprite,CollisionkindList);
```

3. In the system update, check these collision every updated time.

```
@Override
public void update(long elapsedTime) {

    for (Collision collision : myCollisionList) {
        collision.checkGroupCollision();
    }
```

In this case, it is very easy to set collision part. However, the problem is not that simple, we can add some features by the developers. The CollisionKind means the collision between some specific sprites, for example CollisionKindFriend works for two FighterSprites. I gave three kind of very common relations, so if the developers want to detect some strange collision, they can make a subclass of CollisionKind. In order to make this process simple, I already create a template named CollisionKindCustom. The developer might want to create this and setType to get what they want.

```
public void setType(Class<?> c1, Class<?> c2) {
    this.c1 = c1;
    this.c2 = c2;
}
```

In each children class of Reaction, we can manually add some steps that we want to add in this specific reaction in order, then the reaction will do every collision step. It is very easy for developer to add their own steps.

The Reaction is an abstract class. If the developer wants to add some new steps , they can just create some subclasses, and then add into the step list.

```
package PhysicsEngine;
import sprite.SpriteTemplate;

public  abstract class  Reaction {
    protected FightPhysicsEngine myPhysicsEngine;

    public abstract void act(SpriteTemplate ps1, SpriteTemplate ps2);
}
```

Also, I add some new features in this week. One of the most important is gravity.

```
for (FighterSprite sprite : playerSprites){
    MotionAction.Gravity(sprite,1).performAction(elapsedTime);
}
```

Deatails

Physics Engine takes care of the location calculation, and then check whether there is out bound or something else. Then set the sprite into a proper position.

Collision Firstly needs to initialize the collision list, which can be made up of collision within one group or collision between two groups. Every time the system of the game needs to update, then here we need to update all the collisions.

For the collision detection, it is very simple. If there is some sprite not on the right of target, not on the left of target, not on the top of target, not on the bottom of the target, then these two things musk collide.

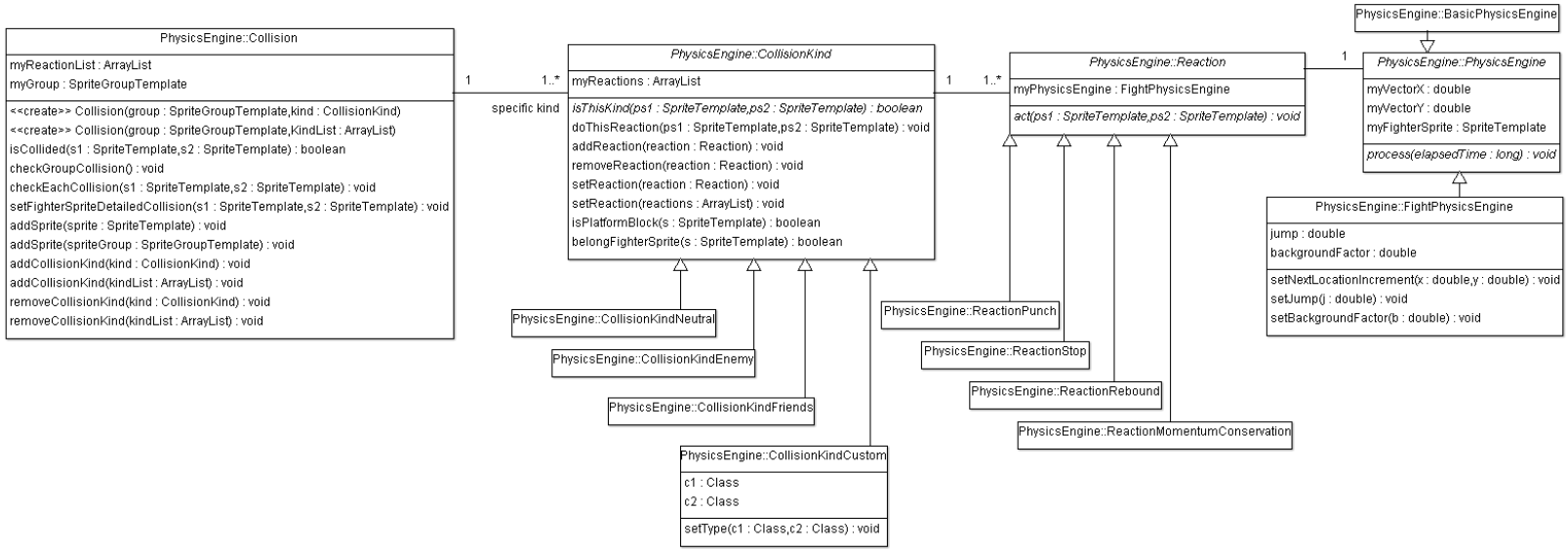
If there is a collision between two sprites, then I use Factory Method Pattern to create a specific collision (the collision between sprite and block, the collision between two enemies, the collision between two friends). Then do the associate reaction.

The reaction is made up of a reaction step list, in this way it is easy to modify the step list. If the programmer wants to add some new features, they can just create some subclass of the step, then add them into the specific list.

Design problem.

For the collision engine itself, I am proud of my design. In my design, developers are very easy to code. I use different of kinds of collision which associate with their reaction to do that, which means it has two levels of hierarchy. The advantage is that developers can put all kinds of sprites into the collision. And in the collision itself, it will find which specific kind of collision it is, and then do the associated reaction. I think another design for this is passing one group in which any pair needs to be checked or two groups between which two sprites from different side into a collision, and we have several collisions to check. Compared to the design I am using, it will run faster, but it needs to code more.

The UML is below.


PhysicsEngine::Collision
myReactionList : ArrayList
myGroup : SpriteGroupTemplate
<<create>> Collision(group : SpriteGroupTemplate,kind : CollisionKind)
<<create>> Collision(group : SpriteGroupTemplate,KindList : ArrayList)
isCollided(s1 : SpriteTemplate,s2 : SpriteTemplate) : boolean
checkGroupCollision() : void
checkEachCollision(s1 : SpriteTemplate,s2 : SpriteTemplate) : void
setFighterSpriteDetailedCollision(s1 : SpriteTemplate,s2 : SpriteTemplate) : void
addSprite(sprite : SpriteTemplate) : void
addSprite(spriteGroup : SpriteGroupTemplate) : void
addCollisionKind(kind : CollisionKind) : void
addCollisionKind(kindList : ArrayList) : void
removeCollisionKind(kind : CollisionKind) : void
removeCollisionKind(kindList : ArrayList) : void
1
1..*
specific kind
PhysicsEngine::CollisionKind
myReactions : ArrayList
isThisKind(ps1 : SpriteTemplate,ps2 : SpriteTemplate) : boolean
doThisReaction(ps1 : SpriteTemplate,ps2 : SpriteTemplate) : void
addReaction(reaction : Reaction) : void
removeReaction(reaction : Reaction) : void
setReaction(reaction : Reaction) : void
setReaction(reactions : ArrayList) : void
isPlatformBlock(s : SpriteTemplate) : boolean
belongFighterSprite(s : SpriteTemplate) : boolean
PhysicsEngine::CollisionKindNeutral
PhysicsEngine::CollisionKindEnemy
PhysicsEngine::CollisionKindFriends
PhysicsEngine::CollisionKindCustom
c1 : Class
c2 : Class
setType(c1 : Class,c2 : Class) : void
1
1..*
PhysicsEngine::Reaction
myPhysicsEngine : FightPhysicsEngine
act(ps1 : SpriteTemplate,ps2 : SpriteTemplate) : void
PhysicsEngine::ReactionPunch
PhysicsEngine::ReactionStop
PhysicsEngine::ReactionRebound
PhysicsEngine::ReactionMomentumConservation
1
PhysicsEngine::BasicPhysicsEngine
PhysicsEngine::PhysicsEngine
myVectorX : double
myVectorY : double
myFighterSprite : SpriteTemplate
process(elapsedTime : long) : void
PhysicsEngine::FightPhysicsEngine
jump : double
backgroundFactor : double
setNextLocationIncrement(x : double,y : double) : void
setJump() : double) : void
setBackgroundFactor(b : double) : void