CPS 108 VOGGA Individual Design

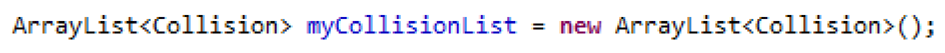
Name: Donghe Zhao
Date: April 13th
Genre: Fighting Game
Teammate: Hareesh, Helena, Chen, Peggy, Wendy, Hui
UTA: Nathan, Tanner
GitLink: https://github.com/hareeshganesan/Vooga/tree/master/src/PhysicsEngine

Introduction

The basic collision process for a game is to create a collision list, then check collision and do some reaction each time the system is updated.

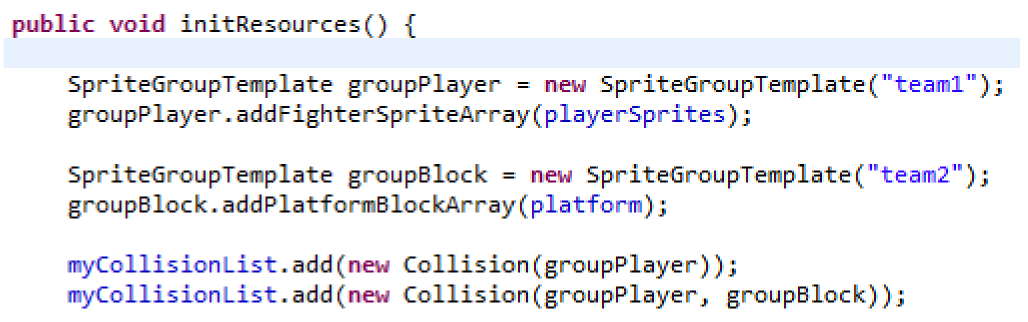
In every game, we just need to initially set the collision with the following examples.

1. Create a instance viable to keep my Collision List



```
ArrayList<Collision> myCollisionList = new ArrayList<Collision>();
```

2. Initial set the collision. I can take care of two kinds of collision. One is passing in one group of sprites, and check any pair within this group and do some reaction. The other is passing in two groups of sprites, then check any one from one group and another one from the other group.



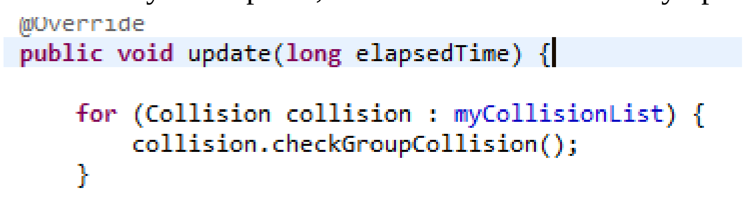
```
public void initResources() {

    SpriteGroupTemplate groupPlayer = new SpriteGroupTemplate("team1");
    groupPlayer.addFighterSpriteArray(playerSprites);

    SpriteGroupTemplate groupBlock = new SpriteGroupTemplate("team2");
    groupBlock.addPlatformBlockArray(platform);

    myCollisionList.add(new Collision(groupPlayer));
    myCollisionList.add(new Collision(groupPlayer, groupBlock));
```

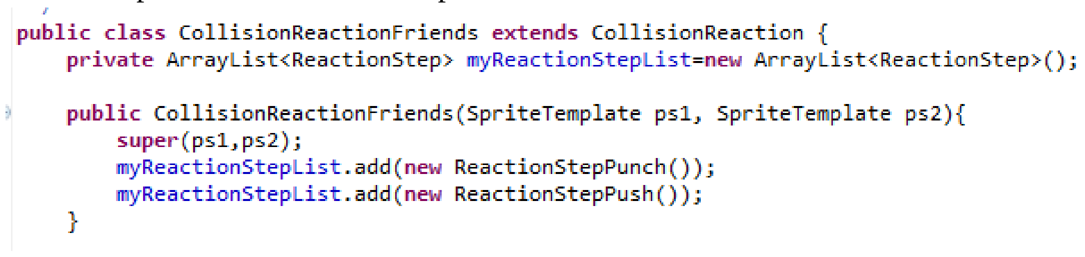
3. In the system update, check these collision every updated time.



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

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

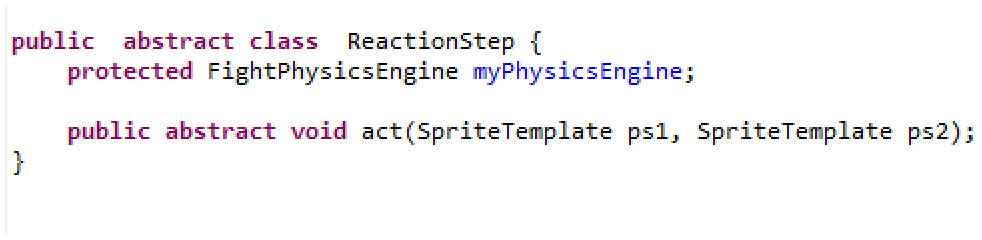
In this case, it is very easy to finish collision part. However, the problem is not that simple, we can add some features by the developers. In each children class of CollisionReaction, we can manually add some steps that we want to add in this specific reaction in order, then the reaction will do every collision step. It is very easy for developer to add their own steps.



```java
public class CollisionReactionFriends extends CollisionReaction {
    private ArrayList<ReactionStep> myReactionStepList=new ArrayList<ReactionStep>();

    public CollisionReactionFriends(SpriteTemplate ps1, SpriteTemplate ps2){
        super(ps1,ps2);
        myReactionStepList.add(new ReactionStepPunch());
        myReactionStepList.add(new ReactionStepPush());
    }
```

The Reaction Step 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.



```java
public  abstract class  ReactionStep {
    protected FightPhysicsEngine myPhysicsEngine;

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

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.

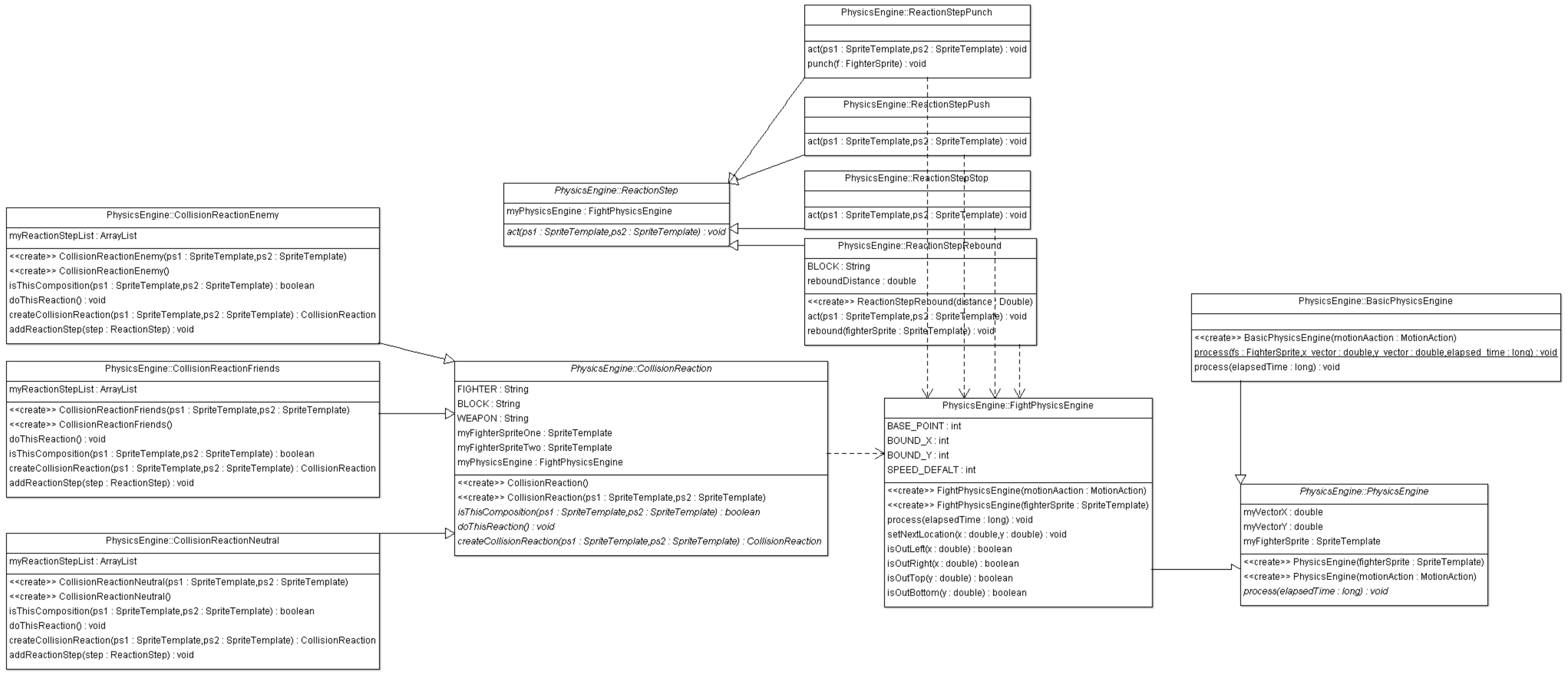
Collision Firstly needs to initialize the collision list, which can be made up of collision within one group or collision between two groups. Every time the system of the game needs to update, then here we need to update all the collisions.

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.

The UML is below.


PhysicsEngine::CollisionReactionEnemy
myReactionStepList : ArrayList
<<create>> CollisionReactionEnemy(ps1 : SpriteTemplate,ps2 : SpriteTemplate)
<<create>> CollisionReactionEnemy()
isThisComposition(ps1 : SpriteTemplate,ps2 : SpriteTemplate) : boolean
doThisReaction() : void
createCollisionReaction(ps1 : SpriteTemplate,ps2 : SpriteTemplate) : CollisionReaction
addReactionStep(step : ReactionStep) : void
PhysicsEngine::CollisionReactionFriends
myReactionStepList : ArrayList
<<create>> CollisionReactionFriends(ps1 : SpriteTemplate,ps2 : SpriteTemplate)
<<create>> CollisionReactionFriends()
doThisReaction() : void
isThisComposition(ps1 : SpriteTemplate,ps2 : SpriteTemplate) : boolean
createCollisionReaction(ps1 : SpriteTemplate,ps2 : SpriteTemplate) : CollisionReaction
addReactionStep(step : ReactionStep) : void
PhysicsEngine::CollisionReactionNeutral
myReactionStepList : ArrayList
<<create>> CollisionReactionNeutral(ps1 : SpriteTemplate,ps2 : SpriteTemplate)
<<create>> CollisionReactionNeutral()
isThisComposition(ps1 : SpriteTemplate,ps2 : SpriteTemplate) : boolean
doThisReaction() : void
createCollisionReaction(ps1 : SpriteTemplate,ps2 : SpriteTemplate) : CollisionReaction
addReactionStep(step : ReactionStep) : void
PhysicsEngine::CollisionReaction
FIGHTER : String
BLOCK : String
WEAPON : String
myFighterSpriteOne : SpriteTemplate
myFighterSpriteTwo : SpriteTemplate
myPhysicsEngine : FightPhysicsEngine
<<create>> CollisionReaction()
<<create>> CollisionReaction(ps1 : SpriteTemplate,ps2 : SpriteTemplate)
isThisComposition(ps1 : SpriteTemplate,ps2 : SpriteTemplate) : boolean
doThisReaction() : void
createCollisionReaction(ps1 : SpriteTemplate,ps2 : SpriteTemplate) : CollisionReaction
PhysicsEngine::ReactionStep
myPhysicsEngine : FightPhysicsEngine
act(ps1 : SpriteTemplate,ps2 : SpriteTemplate) : void
PhysicsEngine::ReactionStepPunch
act(ps1 : SpriteTemplate,ps2 : SpriteTemplate) : void
punch(f : FighterSprite) : void
PhysicsEngine::ReactionStepPush
act(ps1 : SpriteTemplate,ps2 : SpriteTemplate) : void
PhysicsEngine::ReactionStepStop
act(ps1 : SpriteTemplate,ps2 : SpriteTemplate) : void
PhysicsEngine::ReactionStepRebound
BLOCK : String
reboundDistance : double
<<create>> ReactionStepRebound(distance : Double)
act(ps1 : SpriteTemplate,ps2 : SpriteTemplate) : void
rebound(fighterSprite : SpriteTemplate) : void
PhysicsEngine::FightPhysicsEngine
BASE_POINT : int
BOUND_X : int
BOUND_Y : int
SPEED_DEFALT : int
<<create>> FightPhysicsEngine(motionAaction : MotionAction)
<<create>> FightPhysicsEngine(fighterSprite : SpriteTemplate)
process(elapsedTime : long) : void
setNextLocation(x : double,y : double) : void
isOutLeft(x : double) : boolean
isOutRight(x : double) : boolean
isOutTop(y : double) : boolean
isOutBottom(y : double) : boolean
PhysicsEngine::BasicPhysicsEngine
<<create>> BasicPhysicsEngine(motionAaction : MotionAction)
process(fs : FighterSprite,x_vector : double,y_vector : double,elapsed_time : long) : void
process(elapsedTime : long) : void
PhysicsEngine::PhysicsEngine
myVectorX : double
myVectorY : double
myFighterSprite : SpriteTemplate
<<create>> PhysicsEngine(fighterSprite : SpriteTemplate)
<<create>> PhysicsEngine(motionAction : MotionAction)
process(elapsedTime : long) : void