Tank Game Documentation Summer 2021

Simon Wu

920698802

CSC413-01

csc413-su21/csc413-tankgame-swu465: csc413-tankgame-swu465 created by GitHub Classroom

Table of Contents

1 Introduction

This term project is a game with two players that fight against each other using tanks.

1.1 Project Overview

We needed to create two tanks that could shoot each other, where the bullets they shoot could collide into walls, and the enemy tank. We also needed to create three different power-ups that would influence the flow of the game. One power-up can heal you back to full health, another one will give you rockets you can fire that will do twice the damage of a normal tank shell, and the last power-up will give your tank twice the speed.

1.2 Technical Overview

For this project, we were given a skeleton code that we had to fill out the missing pieces until it became a fully functioning game. it was very helpful to use abstractions for creating the necessary classes to build this project. It was recommended to create an abstract class “GameObject” that would be extended by the “Tank”, “Bullet” classes, “Power Up” classes, and the “Wall” classes because many of these classes share the same methods, albeit some minor changes to fit their intended purpose.

By making all of these have the base of “GameObject” I was able to create an ArrayList of GameObjects to make it easier to keep track of everything that would be created and drawn in the main program. The only exception is that the Tank classes have their ArrayList of GameObjects for the bullets that they shoot. This made it easier for me to keep track of which tank was shooting which bullet. To check for the tank, and bullet collision, I created a new class called “CollisionDetection”.

1.3 Summary of Work Completed

Requirement	Fulfillment
Starting screen	
Ending screen	
Two players	
Tanks can move forward and backward	
Tanks can rotate	
Split screen	
Minimap	
Health bars for tanks	
Number of lives for each tank	
3 different power-ups	
Unbreakable walls	
Breakable walls	
Bullets collide with walls	

Bullets collide with tanks	
JAR file in the JAR folder	
README.md filled out	

2 Development Environment

IntelliJ Ultimate Edition

OpenJDK 16: version 16.0.1

3 How to Build/Import your Project

To import the project, “open project” and pick the folder “csc413-tankgame-swu465”. To build the project through IntelliJ, right-click the Launcher.java and pick the “Build Module ‘csc413-tankgameswu465’” option.

To build the JAR file in IntelliJ, go to “Project Structures, under “Project settings” go-to “Artifacts”. Add a JAR file with the option “from Module with Dependencies”. The Module name should be csc413-tankgame-swu465. Pick the Main Class to be “tankrotationexample.Launcher” and hit Ok. Apply and hit OK again to exit the Project Structures screen. Now go to “Build” and go to “Build Artifacts” and “Build”.

4 How to Run your Project

To run the game from IntelliJ, simply right-click the Launcher.java and click on “Run ‘csc413tankgame-swu465’”.

The JAR file to run the project should be in the JAR folder. If you are building the JAR file, it will be in the “csc413-tankgame-swu465/out/artifacts/csc413_tankgame_swu465_jar” folder. To play the game, you can double click the JAR file from IntelliJ, or right-click the JAR file and pick “Run ‘csc413tankgame-swu465’”.

5 Rules and Controls

You cannot phase through walls.

There are two sets of power ups, and they will only spawn once in pre-determined locations.

Normal shoots do 1 damage. Rockets will do 2 damage.

Breakable walls have 2 HP.

Tanks have 4 HP, and up to 3 lives.

Player 1 controls: W and S for movement, A and D for rotating the tank. Spacebar is to shoot.

Player 2 controls: Up and Down arrow keys for movement. Left and Right arrow keys for rotating the tank. Enter key to shoot.

6 Implementation Discussion

GameObject

Abstract methods will be used by most of the classes. Update() and getState() will not be used by every class and will either be blank or return 0. Should be no problems because the classes that don't use getState() never get called.

GameResource

Holds all the BufferedImages that I am using for this game on a Map.

Wall

Specific wall method damaged() is for the BreakableWall class. Because it is void, it is blank for the UnbreakableWall class. This class extends GameObject.

Unbreakable Wall

Holds the coordinates for where the wall exists in the game world, an image of how it looks like, and a Rectangle for its hitbox. The only notable thing this class does is draw the Wall and return its hitbox. Every other function in this class does nothing and will never be used.

Breakable Wall

Like the UnbreakableWall, but the only difference is that it has Hit Points. Once its hit points reach 0, the update function will set the hitbox of this wall to 0 so the tanks and bullets will no longer collide with the wall that has died.

Bullet, Rocket

Holds information necessary for it to travel in a straight line. I followed what was in the video, except taking out the checkborders() function because it was unnecessary since my CollisionDetection class would take care of bullets that hit the Unbreakable walls. **PowerUp**

Extends GameObject and adds its own function pickedUp().

rocketPowerUp, speedPowerUp, HPPowerUp

Like the breakableWall class, it has coordinates, an image, and its hitbox. The hit points, or state, are just an indicator of whether the power-up has been picked up by a tank or not. When it does, the update function inside of each power-up will update its hitbox to disappear from the map.

The rocketPowerUp shoots up to five rockets that will deal two damage instead of one. The speed power-up effectively doubles the speed of the tank. And the HP power-up is just a full heal.

Tank

Like the example in the video with minor modifications. The tank is responsible for activating power-ups that it receives from colliding with a power-up in the game world. It is also responsible for resetting its Hitpoints, and hitbox back to its starting point. If a collision is detected in the CollisionDetection class, it will stop once collisionHappened() is called. The update() function will

keep track of the tank's HP and update its health bar accordingly. It also keeps track of how long the speed boost lasts, and how many rockets have been fired. The tanks will also return X and Y coordinates that enable the TRE class to draw the split screens. The drawImage() will also draw the number of lives left inside the health bar.

CollisionDetection

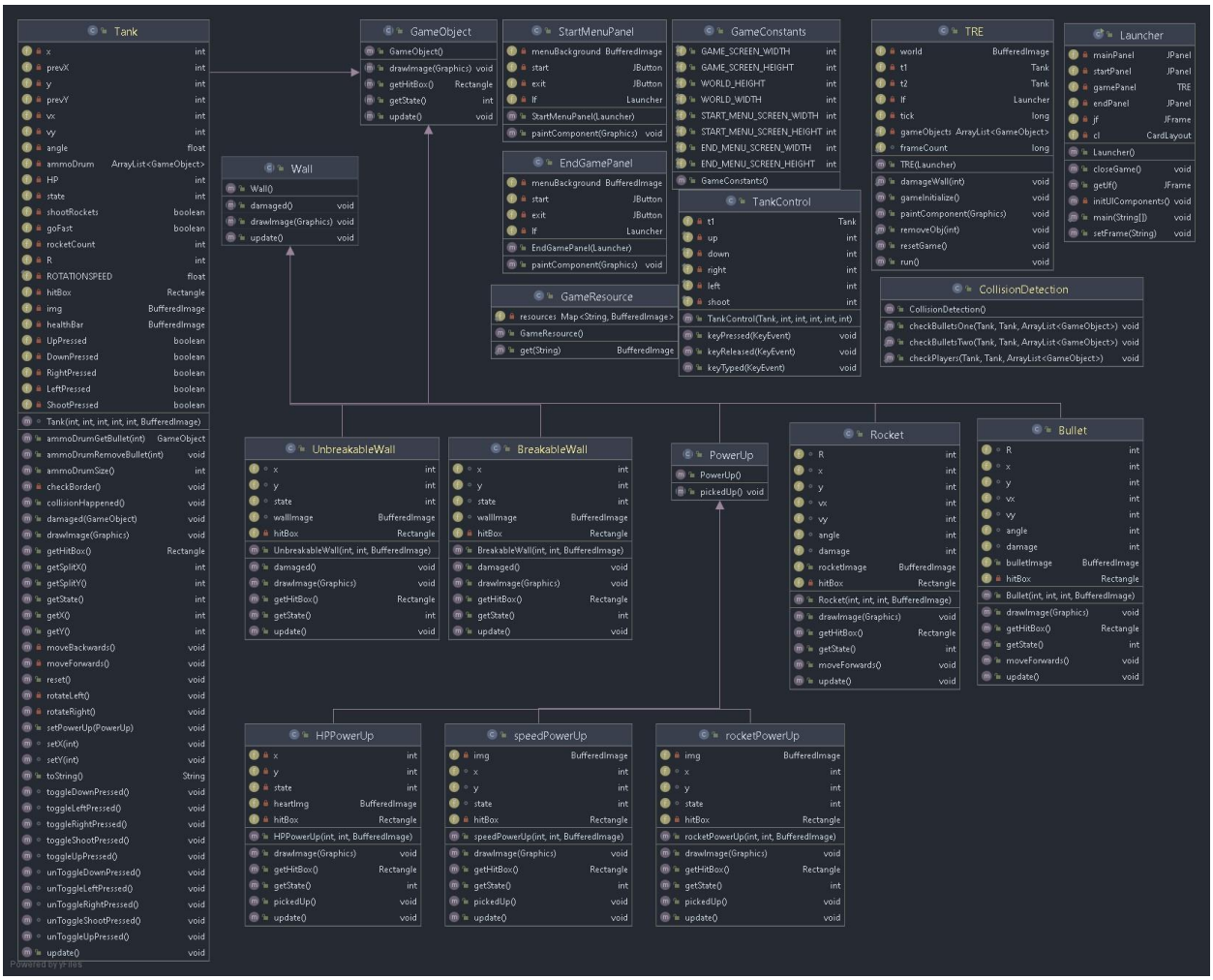
This class has three functions that take in three parameters. Tank 1, Tank 2, and the ArrayList of GameObjects. One function will check both tanks for collisions with walls, power-ups, and each other. The other two functions are for checking the bullet collisions from each tank. One function will check for the bullets from Tank 1, and the other will check for bullets from Tank 2. The bullet collision checks are surrounded by a try-catch in the situation that only 1 bullet exists in the world. Once that bullet hits a wall or tank, an exception will be thrown because there are no more bullets left to check against all the walls and the enemy tank.

If a tank runs into a power-up, this class will call a static function from the TRE class to remove that specific power-up. The same goes for any broken walls.

TRE

Aside from following the video, I made the ArrayList of objects static so that I could create static functions that would be used by the CollisionDetection class. Those functions are removeObj(), and damageWall(). damageWall() will create a new object of BreakableWall from the ArrayList, reduce the HitPoints, and return it to the ArrayList. I am unsure if this is correct, but I am checking for collisions inside of the run() function's while loop. If one of the tanks runs out of hit points, it will turn into the end screen provided.

6.1 Class Diagram



7 Project Reflection

This project had me pulling out my hair trying to figure out bugs. Some of them were very silly and I needed another pair of eyes to look at my code, only to find out it was something simple that I forgot. This project makes me appreciate the polish that I see in good games because of how much work can go into something as simple as a tank war game. But at the same time, I feel like I will be more critical of some of the other games that I play which can have silly bugs. Like how someone fishing in a certain spot in the world can crash the entire server for an MMORPG.

8 Project Conclusion/Results

The game will run and play correctly. Some unexpected exception errors will pop up when playing, but the one that is left does not interrupt the gameplay from the testing that I have done. Overall, I am satisfied that I got this in a working condition that fulfills the requirements for this project. This project also made me very interested in how old curtain fire games are structured, and how does it compare to what I have done for this project.