CSC 413 Project Documentation

Summer 2022

Nicholas Chan

921571713

413.01

https://github.com/csc413-SFSU-Souza/csc413-tankgame-nicknahc

Table of Contents

1 Introduction

1.1 Project Overview

In this project, we are creating a two-player Tank Wars game. This has many aspects to it like coding our tanks and how we control the tanks movement. We also need to code bullets, walls, power-ups and what happens when these things intersect. Next we also code things like health and number of lives. After all that, we code our program to draw images to represent all these things and move them around.

1.2 Technical Overview

In this project, we are creating a Tank Wars game in Java. We must write classes to represent all the different types of entities in our game like Tanks, Walls, Power-Ups, etc.

I wrote an abstract class to represent all of our entities in game. Each Entity has an x, y coordinate (its position), height, width, an image to represent it, and a hitbox. They also need a method getHitBox() in order to process collisions and a draw method to draw our image. This class currently has three child classes, MapEntity, Tank, and Bullet. MapEntity is used to code all the entities that are stationary on the map like walls, breakable walls, and the different power ups.

For our Tanks, we need a Tank class and also a Tank control Class. The TankControl class simply takes in user inputs (our controls for the tank) like up, down, left, right, and shoot. When we initialize our player 1 and player 2 tanks, we can take in the corresponding key bindings as parameters. In addition to our parent data fields, our tank also needs many more like speed, health, lives, rotation speed, angle, etc. Additionally, each Tank has an Array List of bullets that it has fired. When our shoot key is pressed, it creates a new instance of a bullet and adds it to the array list, along with its image, angle, and start position.

For our Bullets, we need an angle, speed, and start position in addition to our parent fields. Additionally, we have an update method that we can call every time we want to update the bullet's position which will change based on its speed and angle.

Next is our GameWorld class which puts all of these entities together and controls the rules of the game. In our class we have an init() method that initializes everything we need for the game. This includes everything like the map, all the images, both tanks, the tankControl instances, and the window that our game will be played on.

Next we have our paintComponent method which uses the Graphics class to draw all our mapEntities, bullets, tanks, and the background.. In addition, this method draws our split screen and our minimap. This also includes any labels on screen that we might want like health/lives count.

Next is our checkCollision method that checks if any hitboxes are intersecting and based on what the entities of those hitboxes are, triggers an event like decreasing health or breaking a wall.

Next is our map maker method which reads our map text file and creates a mapEntity instance for every digit on our map which each digit corresponding to a different type. Most of the other methods are basic graphics except for a couple of game management methods like resetMap or suddenDeath.

We also have a couple of helper classes like Resources and Sound. Resources for the most part stores all of our resources into a Map for easy access, and Sound is used to play our audio files for sound effects/background music.

1.3 Summary of Work Completed

I started with writing the Entity abstract class. From there I coded my Tank, Bullet, and MapEntity which all extend this abstract class. Next I followed the lecture videos to complete the Resource and Sound classes. Next I began writing my GameWorld methods starting with the init() method. From there, I went back and forth with playing with different resources like the map, and the rest of the GameWorld methods. It wasn't until the very end that I implemented the Launcher method which uses the StartMenuPanel and EndGamePanel (which were already given). From there I added certain features like the speedPowerUp and the Sudden Death mode. Towards the end I spent a lot of time cleaning up the code and shortening some bits of code. I didn't necessarily complete all the work in a logical order so I spent a lot of time backtracking and debugging.

2 Development Environment

a. For this project I used Java 17.
b. For this project I used IntelliJ

3 How to Build/Import your Project

Download the project and import it into whatever IDE you are using.

4 How to Run your Project

Once the project is downloaded, run the main function in Launcher.java OR run the jar file in the jar folder.

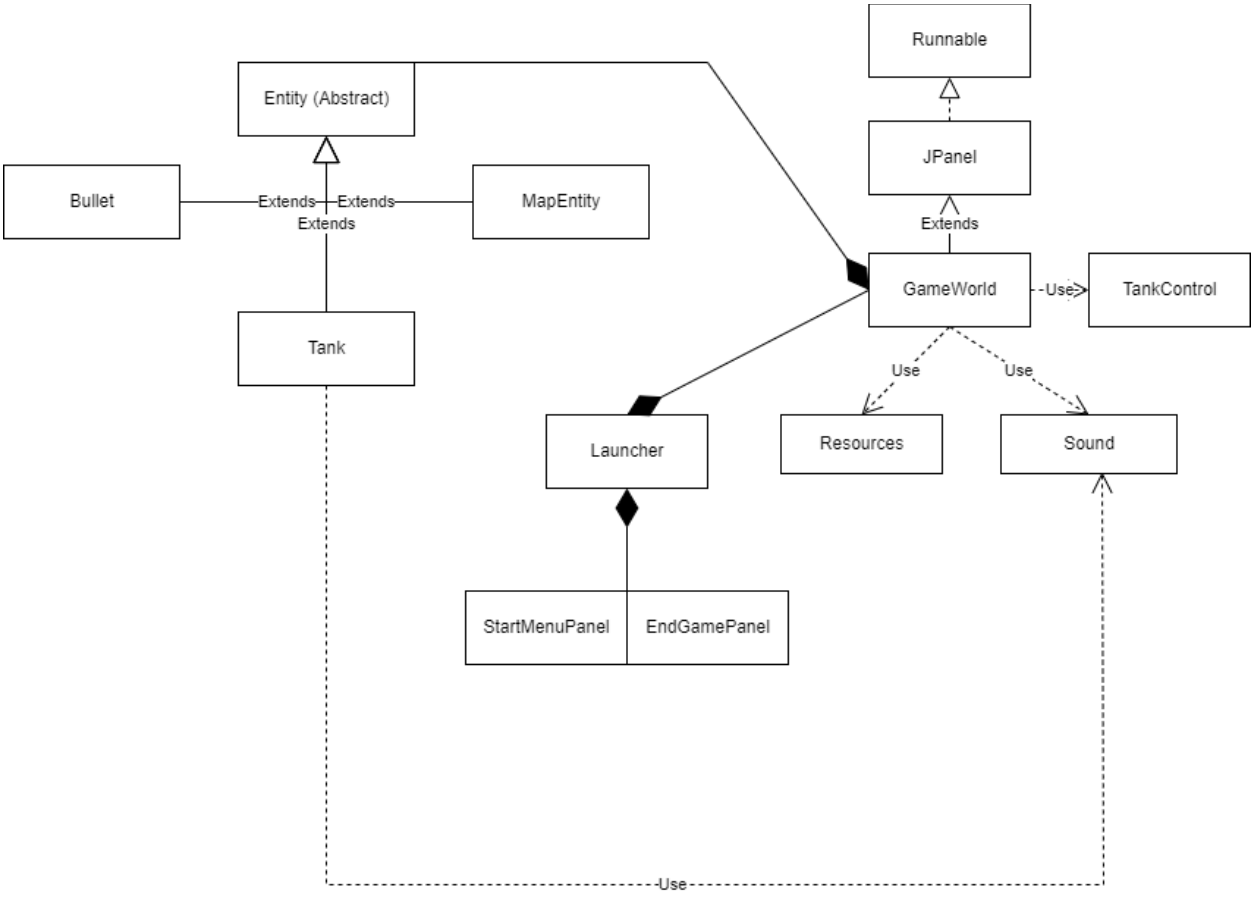
5 Assumptions Made

We can assume that we need a start and end screen. We can also assume that our game must be two-player, with tanks that can move forwards, move backwards, and rotate. Our game must display split screen for each player, a mini-map, health bars, and lives count for each player. We can assume our maps must have unbreakable walls, breakable walls, and three distinct power ups. Our tanks must be able to shoot bullets that collide with walls and other tanks. We can assume that our game must be built into a JAR file and stored into our JAR folder.

6 Implementation Discussion

I changed my program structure a bit compared to my original UML diagram. For my interactive objects in the game, I chose to create an abstract Entity class. Extending this abstract class are three different classes: Bullet, Tank, and MapEntity. Bullet and Tank are self-explanatory, the MapEntity class includes all stationary objects on the map. Unchanged, my GameWorld extends JPanel and implements Runnable. My GameWorld class utilizes my TankControl class and my Resources and Sound classes. My Launcher class simply runs the game through GameWorld and utilizes the StartMenuPanel and EndGamePanel class at the start/end.

6.1 Class Diagram



7 Project Reflection

In my opinion this project wasn't too hard to follow, although the abundance of options in how I could implement my game was daunting at times. I'm pretty satisfied with my finished product and there's not many other features I want to add to my game that I haven't already.

8 Project Conclusion/Results

My project is fully functional and is able to run off the JAR file that I built. Overall, I learned a lot about game design and general concepts. I think it would've been very interesting to see what I could

accomplish with even more time.