CSC 413 Project Documentation

Summer 2021

Aaron Carlson

921241630

CSC 0413-01

https://github.com/csc413-su21/csc413-tankgame-ajccarlson

Table of Contents

1 Introduction 3
1.1 Project Overview 3
1.2 Introduction of the Tank Game 3
2 Development Environment 3
3 How to Build/Import your Project 3
4 How to Run your Project 3
5 Assumptions Made 3
6 Implementation Discussion 4
6.1 Class Diagram 4
6.2 Class Descriptions 5
7 Project Reflection 6
8 Project Conclusion 6

1 Introduction

1.1 Project Overview

On the surface, the goal of the project was to create a fully-function 2D Tank Wars game written in Java. But the real intended purpose of the project was to establish good object-oriented programming (OOP) practices.

1.2 Introduction of the Tank Game

The game is a split screen, player-versus-player tank game where the objective is for one player to deplete all three of the other player's lives by shooting them and avoiding their attacks.

2 Development Environment

I created the project using Java 16 (openjdk-16 version 16.0.2) through the IntelliJ IDEA IDE.

3 How to Build/Import your Project

1. Open command prompt and enter the command "git clone https://github.com/csc413-su21/csc413-tankgame-ajccarlson.git" to clone the repository locally to your computer
2. Open InjelliJ
3. Click on "File", "New", "Project from Existing Sources"
4. Navigate to and select the location of your cloned repository and click "OK"
5. Select "Import project from external model"
6. Select "Maven" and click "Finish"
7. Click on the green hammer icon near the top of the IDE labeled "Build Project"

4 How to Run your Project

1. Open Windows PowerShell or your preferred terminal
2. Enter the command "cd [location to your cloned repository]"
3. Enter the command "cd jar"
4. Enter "java -jar .\csc413-tankgame-ajccarlson.jar"

Alternatively

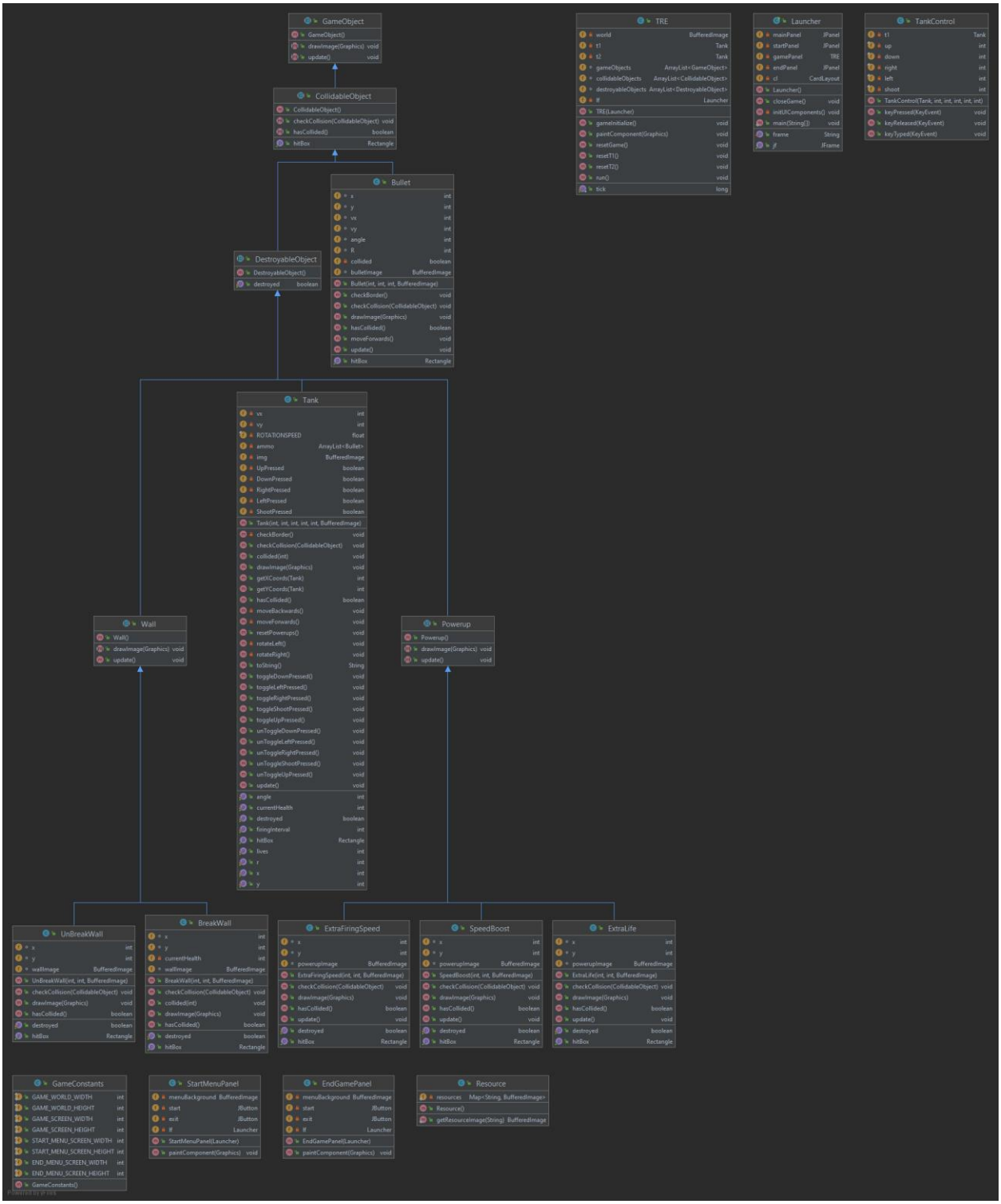
1. Open InjelliJ
2. Open the csc413-tankgame-ajccarlson project
3. Navigate to the jar folder
4. Right click on and run the "csc413-tankgame-ajccarlson.jar" file

5 Assumptions Made

The main assumptions that I made when designing and implementing my project were related to player conduct in the game; namely that the players would actively attack each other since the game only ends when one player loses all their lives and that players wouldn't unleash unending volleys of bullets at the other player's spawn without giving them a chance to fight back.

6 Implementation Discussion

6.1 Class Diagram



6.2 Class Descriptions

I tried to maintain good OOP practices in the creation and design of my classes by setting up umbrella classes such as collidable and destroyable object classes to group together game objects of the same category (including several subclasses branching off of them such as powerup and wall classes acting as abstract classes extending the destroyable objects class), broader packages to contain these classes such as the moveable and stationary packages to contain corresponding objects, and getter and setter functions to practice safe encapsulation.

Now for a quick rundown of each Java class used:

- **dependencies**: Contains classes with values and resources to be referenced by the other game classes.
    - ***GameConstants*:** Contains any fixed constants frequently used by the game. However, it mostly just contains the various game and screen dimensions important to the game.
    - ***Resource***: Contains a HashMap of referenceable resources used by the game, specifically just images in the case of my game.
- **game**: Contains classes directly related to gameplay.
    - **moveable**: Contains object classes that have any type of movement.
        - ***Bullet***: Contains and handles bullet objects shot by each tank through things such as handling the movement of the bullets, creating hitboxes, checking collisions, and drawing the bullets.
        - ***Tank***: Contains and handles the playable tank objects through things such as handling the rotation of the tanks, setting the rate of fire of the tanks, and handling ammo collisions including the detection of and health lost from each shot.
        - ***TankControl***: Contains and handles tank control mappings.
    - **object_classes**: Contains the umbrella classes for game objects.
        - ***CollidableObject***: Abstract class that handles any objects that can collide into one another.
        - ***DestroyableObject***: Abstract class that handles any objects that can be destroyed by tank gunfire.
        - ***GameObject***: Abstract class that handles all interactable game objects.
- **stationary**: Contains object classes with fixed positions.
    - **powerups**: Contains all classes relating to powerups.
        - ***ExtraFiringSpeed***: Powerup class that increases the rate of fire of tanks that pick up the corresponding powerup object.
        - ***ExtraLife***: Powerup class that adds an extra life to tanks that pick up the corresponding powerup object.
        - ***Powerup***: Abstract class that handles all powerup objects.
        - ***SpeedBoost***: Powerup class that increases the movement speed of tanks that pick up the corresponding powerup object.
    - **walls**: Contains object classes related to wall objects.
        - ***BreakWall***: Contains and handles breakable wall objects on the map through things such as checking for collisions, checking the health of the object, and marking the wall as destroyed.

        - ***UnBreakWall***: Contains and handles unbreakable wall objects on the map through the same means as the BreakWall class without anything related to the walls' destruction.
        - ***Wall***: Abstract class that handles all wall objects.
- **TRE**: The class that handles the overall execution of the game; initializing the game, loading the map, assigning objects to umbrella classes, handling the destruction of objects, rendering the game, etc.
- **menus**:
    - ***EndGamePanel***: JPanel class that handles everything related to the end game screen such as displaying it, creating and listening to the interaction with buttons, and executing the restart and exiting of the game depending on which button is pressed.
    - ***StartMenuPanel***: JPanel class that handles everything related to the start game screen in the same manner as EndGamePanel, but with different images and with a button option relating to starting the game instead of restarting it.
- **Launcher**: The main class of the project. Mostly handles the creation and framing of the various JPanels used for the game's menus as well as ensuring that the game launches and exits properly.

7 Project Reflection

I found the project to be incredibly difficult at first since I had no experience in coding for or designing games, but the further I progressed into the project, the more comfortable I got into the coding, and the more fulfilling I found it to be. The project required me to use and be aware of a great variety of different coding and non-coding skills, which I found to be a great way to practice everything that I've learned about Java so far. Lastly, I'm very glad I started working on the project early, since it gave me a lot of time to work things out and experiment instead of having to worry about how to make sense of and implement a lot of new concepts right before the deadline, which would've led to a lot of unnecessary stress. Overall, though the work on the project was a bit frustrating at times, it's something I'm very glad I worked on and proud in having made.

8 Project Conclusion

Though I'm very proud of how my term project turned out, I feel that given a little more time and by utilizing the new experience I gained through my current implementation of it, I could create a much better and more engaging tank game. Some things that I would've liked to include in my project that I passed on including due to lack of time or understanding include BGM for the menu and game itself to make it a little more pleasant to play, SFX for object collisions and destructions, explosion animations for bullet collisions and object destructions, a score counter for kills scored by each player, and either maps that would auto generate something new with each round or select from a handful of premade maps/environments.