CSC 413 Final Project Documentation

Summer 2021

Olivier Chan Sion Moy

913202698

413.01

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

Table of Contents

1 Introduction

1.1 Project Overview

This project is a 2D game programmed in Java. Classes are programmed with an OOP paradigm.

Game graphics are rendered using JPanels and JFrames.

The game supports screen resizing, however the code and compiled jar have hard-coded resolutions.

In-game movement, animations, and other events are independent of the game's framerate.

1.2 Tank Game

The game is a two-player competitive game where both players control tanks that shoot and can pick up and use power-ups. The objective of the game is to outmaneuver your opponent and utilize powerups and the environment to destroy your opponent's tank 3 times to win.

The environment consists of walls and power-ups.

There are two types of walls - unbreakable walls, and breakable walls. Breakable walls will be destroyed after being sufficiently damaged by tank shots, whereas unbreakable walls cannot be destroyed.

There are four types of power-ups:

- Health: Restores all tank health.
- Speed: Temporarily increases tank movement speed.
- Weapon Upgrade: Temporarily fire 3 bullets in an arc instead of 1 bullet.
- Shield: Temporarily become invulnerable, unable to take damage from bullets. Getting hit reduces the duration of the shield.

2 Development Environment

IDE:

IntelliJ IDEA 2021.1.3 (Ultimate Edition)

Build #IU-211.7628.21, built on June 29, 2021

Runtime version: 11.0.11+9-b1341.60 amd64

VM: OpenJDK 64-Bit Server VM by JetBrains s.r.o.

Kotlin: 211-1.4.32-release-IJ7628.19

Java Version Used:

openjdk-16 version 16.0.2

Language level: 8

3 How to Build/Import your Project

Instructions given are for the aforementioned development environment.

1. Click "File->New->Project From Version Control"
2. Use "https://github.com/csc413-su21/csc413-tankgame-oliviercm.git" as the URL.
3. Click "File->Project Structure"
4. Set "Project SDK" to openjdk-16 version 16.0.2. You may have to download the SDK through "Add SDK->Download JDK".
5. Set "Project Language Level" to "8 - Lambdas, type annotations etc.".
6. Click "Modules".
7. Right-click the "resources" folder and select "Resources".
8. Press Alt+F9 to build the project.
9. To create a JAR, click “Build->Build Artifacts->csc413-tankgame-oliviercm:jar->Build”.

4 How to Run your Project

1. Follow all instructions in Section 3.
2. Right-click /src/tankrotationexample/Launcher in the Project Overview tab and select “Run”.

In the window that pops up, click “Start” to begin the game.
Controls:
W/A/S/D - Left player movement
Space - Left player shoot

Up Arrow/Left Arrow/Down Arrow/Right Arrow - Right player movement

Enter - Right player shoot

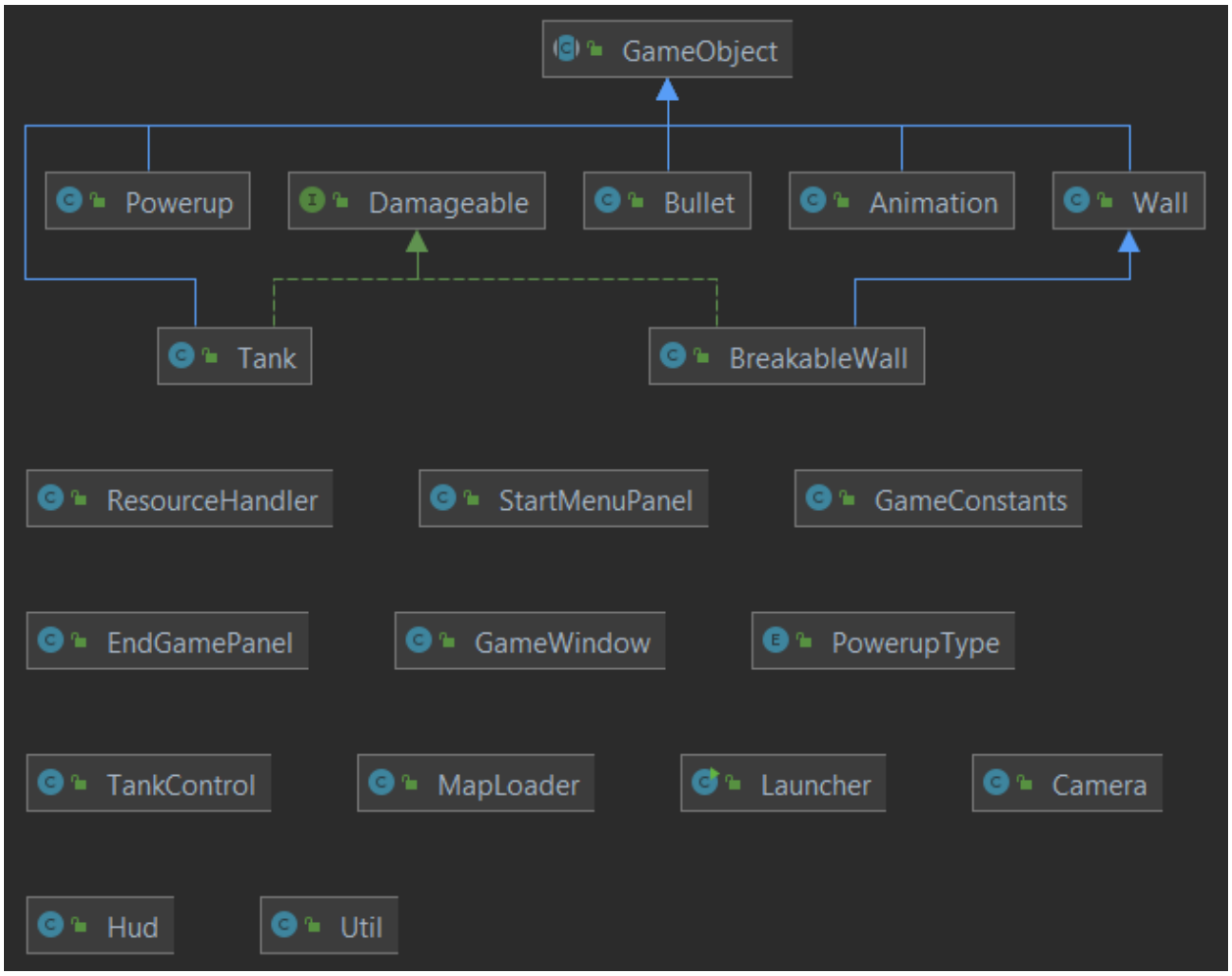
The objective is to destroy the opponent’s tank 3 times. Various powerups and the environment can be used to outmaneuver and outplay your opponent.

5 Assumptions Made

It is assumed that:

- The running computer has a screen that is suitable for the hardcoded resolution of the game.
- The running computer has arrow keys.
- The running computer has a compatible version of Java installed to run the JAR.
- At times, it is assumed that there are only 2 players (although sometimes it is not, and a dynamic amount of players is supported).

6 Class Diagram



7 Class Descriptions

7.1 Launcher

- Contains driver code which starts the program and instantiates the start/end screen, and the main game screen.

7.2 GameWindow

- The handler of the main game's screen (graphics) and gameplay loop.
- Responsible for starting the game by calling helpers such as MapLoader/ResourceLoader, and instantiating players (Tanks) into the game world.
- Contains the gameplay loop.
    - Each iteration of the gameplay loop calls "update()" on all Gameobjects, while also providing the amount of elapsed time since the last iteration, allowing GameObjects to adjust their behavior based on real-world time instead of being framerate-locked.
    - Each iteration of the gameplay loop draws and renders the game world and various display elements by creating a blank image as a buffer and having all GameObjects draw themselves onto the buffer, then retrieving a proper subimage of the world centered

around the players. The subimages are then rendered on screen, with the addition of elements such as a minimap and player lives count.

7.3 GameConstants

- Static class containing various constants such as window size and game world size.

7.4 StartMenuPanel, EndMenuPanel

- Simple classes used to draw and implement the start and end screens, including instantiating clickable buttons.

7.5 GameObject

- Abstract parent class of all in-game entities, such as tanks, bullets, walls, and animations.
- All GameObjects are tracked by a static data structure, and are added and removed on construction and deletion.
    - GameObjects are removed from the world by calling a cleanup method “destruct()”. This method also removes the GameObject from the static data structure.
- Have a position on a 2D plane consisting of a coordinate pair.
- Have an angle representing a direction on a 2D plane - a.k.a. rotation.
- Have a bounding box (a.k.a. hitbox) defined by a coordinate pair. The coordinate pair represents an offset from the Object’s position, thereby defining the bounding box as the rectangle such that the top-left corner is the Object’s position, and the bottom-right corner is the point which is Object’s position offset by the bounding box coordinate pair.
- Provides utility method to retrieve all GameObjects with a bounding box intersecting the context object’s bounding box.
- Provides utility method to translate the GameObject’s position a certain distance based the the GameObject’s rotation (angle).
    - An alternate method is also provided that also checks for and prevents translating into a solid object by instead “sliding” alongside the object.
- Provides utility method to draw graphics onto the game world.

7.6 Damageable

- Interface.
- Classes implementing Damageable must have a “health” value, and must implement a “takeDamage()” method.

7.7 Tank

- Extends GameObject.
- Implements Damageable.
- Represents the object controlled by players (the tank).
- Contains core gameplay elements including health, lives, movement speed, and rotation speed.
- Handles player input by modifying the world/tank (moving the tank forwards and backwards, creating bullets, picking up powerups, etc.).
- Implements taking damage by reducing health. Once health reaches 0, the tank is destroyed and is respawned shortly after if the player has extra lives. Otherwise, the tank sets a game over state.
- Picks up powerups - powerups then modify the tank such as increasing the tank’s health, making the tank faster, etc.

7.8 Bullet

- Extends GameObject.
- Instantiated by Tanks when firing. The firing Tank is designated as the “owner” of the Bullet.
- Upon collision with a solid GameObject, if the GameObject is not the Bullet’s owner (the tank that fired the Bullet), the Bullet will “explode” and disappear. Additionally, if the colliding GameObject implements Damageable, that object will take damage.

7.9 Powerup

- Extends GameObject.
- Created as a map element.
- Provides a method which, based on the powerup’s type, modifies a Tank, such as increasing health, speed, etc.

7.10 Animation

- Extends GameObject.
- Non-solid (does not block Tanks, Bullets, or other movement).
- Using a spritestrip, Animations change their image based on a given amount of frames and framerate, by cutting subimages from the spritestrip.
    - Upon playing every “frame” from the spritestrip, automatically destructs itself.

7.11 Wall

- Extends GameObject.
- For use in creating an interesting environment to play in - Walls are impassible by Tanks during movement and also block Bullets.

7.12 BreakableWall

- Extends Wall.
- For use in creating an interesting environment to play in - BreakableWalls are impassible by Tanks during movement and also block Bullets.
    - After blocking enough Bullets, the Breakable wall runs out of health and is destroyed (disappearing from the game world).

7.13 Camera

- Helper class, storing information such as a GameObject to “follow” (Tanks) and a position on a grid (split screen grid).

7.14 Hud

- Helper class, providing methods to draw screenspace and worldspace elements such as health bars and lives on screen.

7.15 MapLoader

- Helper class, providing method to read map file from resources and instantiate Walls and Powerups based on the map file contents.

7.16 ResourceHandler

- Helper class, providing methods to load resources (images), and accessors to retrieve images based on arbitrary keys.

7.17 Tank Control

- Helper class, taking key inputs and calling relevant methods on its attached Tank Object.

8 Self Reflection on Development Process

Overall, the project was not overly difficult. This project was not the first game I have created, so the concepts were not difficult to grasp. I mostly learned about various parts of the Java standard library, and also gained some insight into the runtime/execution of Java due to a race condition that I had to debug during development.

9 Project Conclusion/Results

In conclusion, the project was successful, and accomplished the given requirements.