1. Introduction

Project Overview

The Tank game is an ideal term project because it entails both designing and implementing a fascinating game from start to finish.. The main goal of the game is to achieve fast, challenging gameplay for two players where both of them control tanks and the field is full of barriers. This project also provides real-world ideas related to object-oriented programming and game development as well as resource optimization and use of libraries. It is a focal point of competitive play where the two players are engaged in an ongoing fight.

Introduction of the Tank Game

The Tank game is a 2D arcade shooting game in which players can enjoy the real intense battle experience. Players control a tank, moving it across a map and fighting against the opposite tank. The goal is to fire at the enemy tank and do not allow them to shoot back at you. The game ends once either tank loses their lives completely. Through this mechanism, gameplay remains stimulating and rewarding for the player in the process of completing a level.

2. Development Environment

Version of Java Used

Java SE Development Kit 8u271

IDE Used

IntelliJ IDEA Community Edition 2020.3

Special Libraries and Resources

- **LibGDX** for game development: LibGDX
- **Tiled** for map creation: Tiled Map Editor

3. How to Build or Import Your Game in the IDE

Importing the Project

Download and Install IntelliJ IDEA:

- If you haven't already, download and install IntelliJ IDEA

Clone the Project from GitHub:

- Open terminal or Git Bash.
- Navigate to the directory where you want to clone the project (e.g., `cd Documents`).
- Clone the repository using the following command: git clone

Open IntelliJ IDEA:

- Launch IntelliJ IDEA

Import the Project:

- Click on `New Project from Existing Sources`.
- Navigate to the cloned project directory
- Select the project folder and click `OK`.

Wait for IntelliJ to Set Up the Project:

- IntelliJ IDEA will index the project and download any necessary dependencies.

Building the JAR

Open Your Project:

- Launch IntelliJ IDEA and open the Tank Game project.

Open Project Structure:

- Go to `File > Project Structure`.

Configure Artifacts:

- In the Project Structure dialog, select `Artifacts` from the left-hand menu.
- Click the `+` button to add a new artifact.
- Select `JAR > From modules with dependencies`.

Setup the JAR Configuration:

- In the dialog that appears, select your main module (e.g., `tankgamepack`).
- Make sure to specify the Main-Class (e.g., `tankgamepack.Launcher`).
- Click `OK`.

Include Resources:

- Ensure that any resources (like images, sounds, etc.) that your application needs are included in the JAR.
- We can do this by adding them to the `Available Elements` section in the `Output Layout` tab.

Build the JAR:

- Once the artifact is set up, go to `Build > Build Artifacts`.
- Select the artifact you just created (e.g., `tank gamepack:jar`) and click `Build`.
- IntelliJ IDEA will build the JAR file and place it in the `out/artifacts` directory of your project.

Running the Built JAR

Navigate to the Directory:

- Open a terminal or command prompt.
- Navigate to the directory where your JAR file is located. This is typically in the `out/artifacts` directory of your project."in my case" (cd path/to/your/project/out/artifacts

Run the JAR File:

- Use the `java -jar` command to run your JAR file. Replace `yourjarfile.jar` with the name of your JAR file. (java -jar yourjarfile.jar)

4. How to Run Your Game

Running the Game

Open IntelliJ IDEA:

- Launch IntelliJ IDEA from your desktop or start menu.

Open Your Project:

- Go to `File` > `Open`.
- Navigate to the directory containing the Tank Game project and select it.
- Click `OK` to open the project.

Navigate to the Main Class:

- In the project window, navigate to `src/tankgamepack/Launcher.java`.

Run the Main Class:

- Right-click on `Launcher.java` and select `Run 'Launcher.main()'`.
- IntelliJ IDEA will compile and run the project, launching the Tank Game.

Game Rules and Controls

Blue Tank Controls

- **Move Forward:** Up Arrow (↑)
- **Move Backward:** Down Arrow (↓)
- **Rotate Left:** Left Arrow (←)
- **Rotate Right:** Right Arrow (→)
- **Fire:** Enter Key (`Enter`)

Red Tank Controls

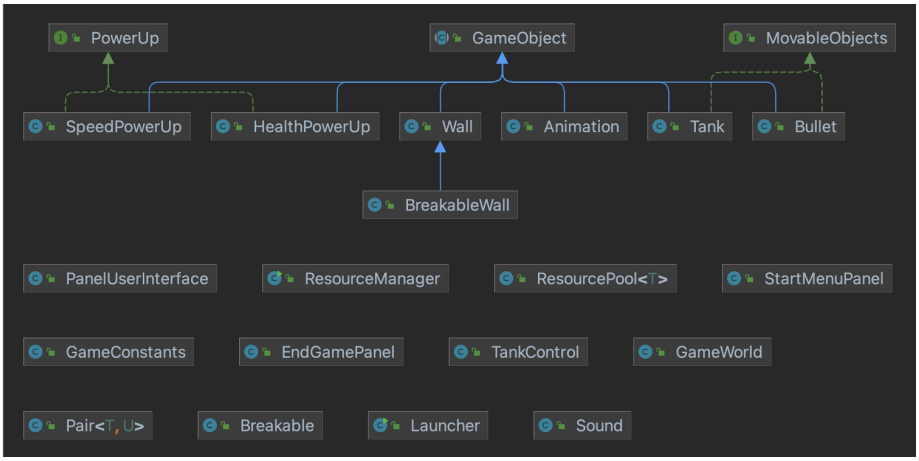
- **Move Forward:** `W` key
- **Move Backward:** `S` key
- **Rotate Left:** `A` key
- **Rotate Right:** `D` key
- **Fire:** Space Bar (`Space`)

The general objective of the Tank Game is to destroy the rival tank. In the game there are two players who each has one tank on a territory consisted of some barriers such as walls. The first one to destroy the tank of the opponent is the winner of the game. Navigate your tank in a specific position on the map and manage the firing system in an attempt to eliminate the enemy tank while ensuring that your own tank is not destroyed.

5. Assumptions Made

- The player is familiar with basic keyboard controls.
- The game is intended to be a multi-player experience.
- The game runs on any platform that supports Java.

6.Tank Game Class Diagram



7. Class Descriptions

Launcher

Handles different panels of the game such as start menu game and end game through CardLayout and sets up the main and graphical user interface of the game and sets the game to run in a separate thread.

GameConstants

Specifies parameters that remain fixed throughout the game; some of these are the screen sizes for the different panels that are present in the game – the start-up menu, the game layout, and the end menu.

ResourcePool

Coordinates the usage of a given resource (s, for instance, bullets) to enhance the utility of a resource. Can add a new resource to the pooling, perform get and put operations on the pool, and put a resource back to the pool.

ResourceManager

Responsible for loading and managing the game resources including but not limited to sprites, animations, and sounds. Instantiates these resources and gives access to these resources.

Pair

A base class of a type that represents a pair of objects. Used to store and manage pairs of values.

StartMenuPanel

Specifies the panel where the start menu of the game is created. Creates the environment, the buttons ‘Start’ and ‘Exit’ and their functionalities.

PanelUserInterface

Holds functions that are used for formatting buttons that are used in the game’s interface panels.

EndGamePanel

Describes the end game panel of the game. Creates the name window, the buttons: Restart Game and Exit, and what they should do when pressed by the user.

Wall

Defines a wall object of the game, wherein contains a position, an image, and a hit-box. Can draw itself meaning that it can draw the next frame and has functionality to handle collision.

TankControl

Implements KeyListener to listen for key events to control a tank. Cores the key presses and releases for the tank’s movement and to make the tank shoot.

Tank

Player class of the game that provides attributes for position, speed, direction of the tank, and has a health parameter. Is responsible for moving, shooting, detecting if a collision has occurred and for drawing itself as well. It is in charge of power-ups and updates its state depending on the user's feedback.

SpeedPowerUp

Represents a speed power-up in the game. Improves the tank's movement speed when it has collected them. Contains draw functions that relate to itself, collision detection, and to check if it is alive or not.

Sound

Works with the sounds that can be heard within the frame of the game. A wrapper to a Clip object and allows the sounds in them to be played, stopped, looped or the volume level adjusted.

PowerUp

A set of controls that affect the game's elements. Describes how to determine if a power-up is no longer valid and if it is still active in addition to defining the activation health.

MovableObjects

A class for the set of objects that can move in the game. Prescribes ways to modify the object's state and determine its freshness.

HealthPowerUp

Another sign which depends on the game is that it represents a health power-up in the game. Raises the health of the tank if it collects/earns them. Contains procedures for rendering itself, for collision detection, and for determining whether or not it is alive.

GameWorld

Responsible for the general supervision of the game world including the game loop, update of objects, detection of collisions, and the rendering of the game world. It is the first and root class for the game, in charge of its initiation and managing of game resources.

GameObject

An abstract class that defines a base class for a general form of a game object. It covers techniques for the location of the object as well as techniques for obtaining a hitbox and methods of handling collisions. It also contains a static method that allows creating new instances of some objects that belong to the game according to the type.

Bullet

Represents a bullet in the game, has position, velocity, angle, and owner. Controls the movement of the object, checks for the collisions, draws the object itself, as well as plays explosion animation when the object collides with anything.

BreakableWall

Serves as a symbol of a wall in the game which can be destroyed. The Wall class contains methods that update the image when it is broken and the check if it is broken.

Breakable

A blank class, likely to be used for future updates or serves as a flag.

Animation

Stands for an animation of the playing game. These are the characteristics for position, frames, time between frames, and the fact whether the given animation is working. Extracts fresh contents of the current frame and paints the animation.

8. Self-Reflection on Development Process

Overall, I got very useful experience of game development using Java during the entire term project. It served my purpose of knowing how to structure game loops, dealing and reading raw user inputs, and understanding the gameplay mechanics of a basic multiplayer tank game. There were difficulties with performance, such as working with memory, figuring out animations, and determining the best way to quickly render the game, thus I had difficulties with performance optimization and trying to make other animations appear in the game. One of the biggest difficulties that were encountered was maintaining proper flow during gameplay while having to deal with movements of multiple objects at the same time and their animations. In general, this project helped me improve my problem-solving abilities and expand my ideas and knowledge concerning games and Object-Oriented Programming.

9. Project Conclusion

The Tank game project encompassed most of the necessary activities in the software development life cycle, which included; design, implementation, testing and optimization. It was beneficial for the practical application of the theories learned in a real-life situation and to create a functional multiplayer game from the ground up. The final product is a game that can be played by someone and essentially shows elementary concepts of game design and Object Oriented Programming. This project has been useful where we have been able to gain experience that will be used in future projects in software development and game designing.