CSC 413 Project Documentation

Spring 2021

Tank Game

Kyle Harvey

915139815

CSC 413-01

https://github.com/csc413-su21/csc413-tankgame-k-harvey.git

Table of Contents

1 Introduction

1.1 Project Overview

A multiplayer split screen tank game. Two users can play different tanks in two views. Each players goal is to shooting enemy tank until the health of the enemy tank is zero.

1.2 Technical Overview

For this project, the goal is build a tank game where the two tanks will battle each other until one is destroyed. There will be two split screen windows where two users can play as different tanks. Each tanks have a health bar and 3 lives, they must pick up 3 of the power ups available that will boost their chances of winning. The tanks will fire a bullet that will shoot to take out the other tank. Also have a collision detection for all the ticking objects and game objects.

2 Development Environment

Version of Java Used: Java 16

IDE Used: InteliJ IDEA 2021.1.2 Ultimate Edition

Resources: Downloaded sprites from google image searches.

3 How to Build/Import your Project

In order to import this tank game project as a JAR file, first go into project structure and select artifacts. We add the out directory that we want to make a JAR which should just show the project you have open and the main class that's running the game, after selecting hit apply and OK. After that we could and build artifacts with it inside the out folder. It'll produce a JAR file zip that could be moved into another folder and open the game by right clicking to open and run without needing to build through the compiler.

4 How to Run your Project

The project runs when you click run after building the project to make sure it compiles properly with no errors. Once you're able to compile with no issues, you could run the main program and that will start the tank game. The process should look like this:

Run Intellij

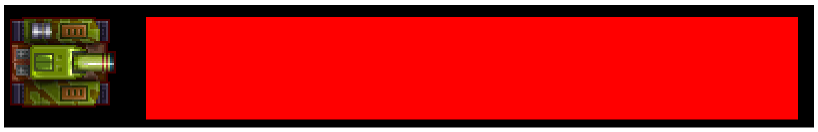
IntelliJ : File -> New -> Project from Existing Sources -> Browse to the project folder and open all -> tankgame . To make the game run, you must run **TRE** class.

JAR : jar -> csc413-tankgame-k-harvey.jar -> right click -> Run

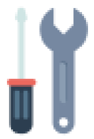
Rules and controls:

	Player 1 Keys:	Player 2 Keys:
Turn Left:	**A**	**Left Arrow**
Turn Right:	**D**	**Right Arrow**
Forward:	**W**	**Up Arrow**
Backward:	**S**	**Down Arrow**
Shoot:	**Space**	**Enter**

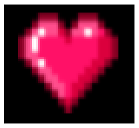
Health: How much health the players has left. Health power ups will replenish any lost health.



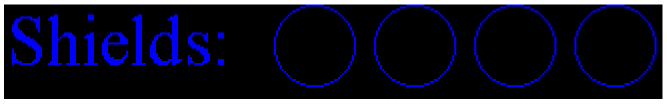
Health Power up:



Lives: How much lives a player has left, drops down once a health bar is depleted

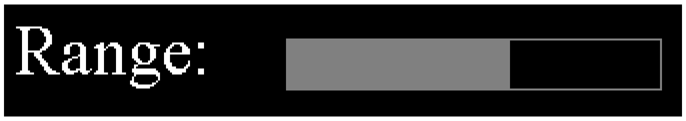


Shields: Shows you how many shield power ups you picked up, protects you from taking damage from bullets.



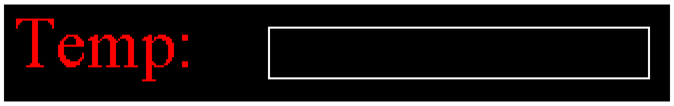
Shield Power up:

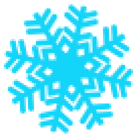
Range: Shows how much range your bullets have, the more range power ups you pick up the further bullets will travel.



Range Power up:

Temp: Your temperature gauge for your bullets fired, the more bullets fired the hotter it gets and when fired too many will be needing to cool down. Temp power up will lower the cool down.





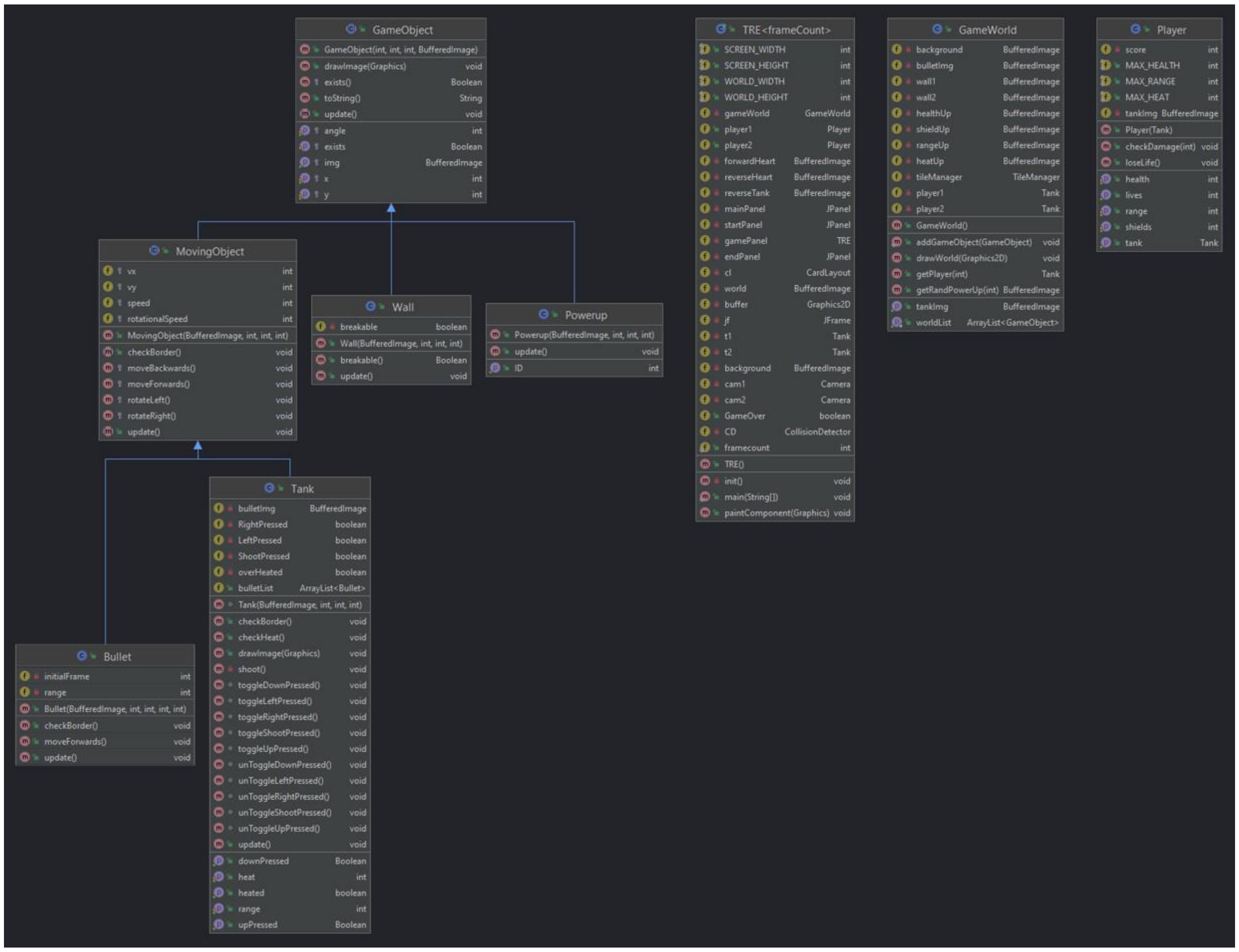
5 Assumption Made

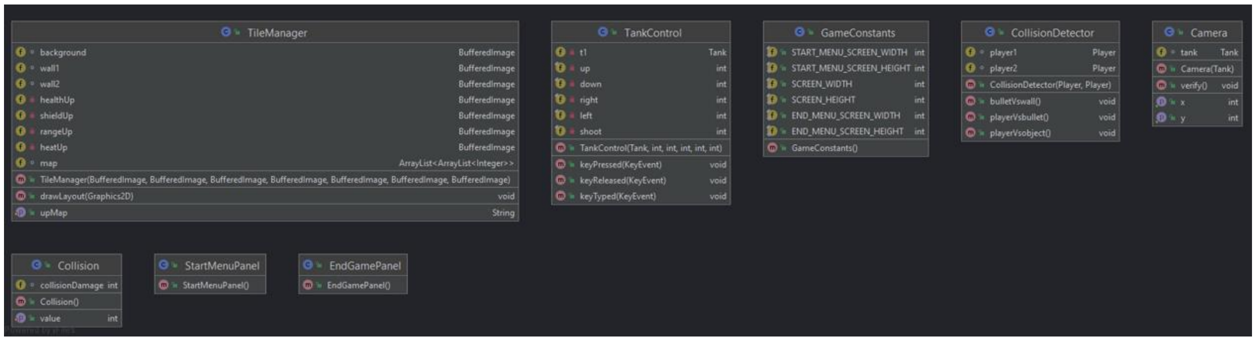
Some assumption made when it came to designing this project that it would be easy to follow from the videos alone but that was not the case. Most of the design and implementation portion came from the guidelines, such as the implantation containing a splitcreen and minimap. The game also requires a moving object, bullet, and tank. Within the game we needed to make a map that has walls both unbreakable and breakable when bullets hit it, which is the collision logic. Once we covered most of the fundamental parts of the game, the rest comes down to design process for the game how we want it to be played.

6 Implementation Discussion

For implementing this project, we were provided with starter codes to help give that push which provided a Tank, a Tank Control, and TRE to put the Tank and game together. There's also menus for the Start Menus and End Game panel that uses the GameConstants class, all of it could be ran with the Launcher class. Although I didn't exactly use all the starter codes, I was able to work with some of it such as Tank and Tank control to make other Java files such as Player, the Camera following to show the Tank along with the camera for the minimap. Also needed to make a gameobject class for all the objects that we would be using such as the walls, tanks, powerups, and bullet. Once we have our objects created, we could focus on collisions for the tanks and bullets so we have both the collision detector class to take all the bullets from both the players and the damage if the bullet object enters the tank's hitbox. Using the Wall class and power ups, we use the Tile Manager class to manage those objects to put our maps together after we design it in a text file using excel tables. Once we have all the java files needed for the game, we could focus on the TRE which is where main is for putting together everything and creating the game when we compile and run it.

7 Tank Game Class Diagram





8 Class Descriptions of classes implemented in the Tank Game

Bullet – The bullet class extends MovingObject. This class is one of the moving object that is continuously render until it hits the wall. The class contains constructor that holds the location and image of the bullet. This class also contains the checkBorder() method which provide a way to detect where the tank is facing. The class also contains an moveForwards() method which detects the collision of bullet and wall. It is use to play the sound of bullet hitting the wall and removing bullet when it hits the wall.

Camera- The Camera class contains the logic behind the tank player class. It is the way to detect whenever the tank and moving within the screen and follow it.

CollisionDetector- The CollisionDector class keeps tracks of all the collision that happens to both the players and the walls. We have the playerVsbullet() method to know when a player gets hit by a bullet taking damage. The bulletVswall() method shows what happens when a bullet hits a breakable wall and a unbreakable wall. The playerVsobject() method is for when players run into walls or powerups.

EndGamePanel- The EndGamePanel class was created to for the end of the game but was not used to having enough time to make it work.

GameConstants- This GameConstants class was created to hold values for the start menu, screen for the game, and the menu when it ends but was not used.

GameObject- The GameObject class detects all the game objects location position x and y.

GameWorld- The GameWorld class contains all the images that will be use as a sprite in the game and where the tank animation of multiple images will be buffered using different rows and columns on the sprites.

MovingObject- The MovingObject class extends GameObject to see where objects are moving. It contains a MovingObject, moveForwards, rotateLeft, rotateRight, and moveBackwards methods while checking the borders and updating it.

Player- The Player class contains the animations that will be animated in the game. We use the animation class to animate the tank moving. We use the render method in Player the render the animation of the tank. The update methods contain the logic which is used to detect collision between both tanks.

Powerup- The Powerup class extends GameObject for the power ups in the game, it checks for the image on what it is and where it's placed, which used for the Tile Manager.

StartMenuPanel- The StartMenuPanel class was created to for the start of the game menu prompting players to start the game or exit but was not used to having enough time to make it work.

Tank- The Tank class extends MovingObject to create the player's tanks and the controls on what they do in the game. Includes updating all its movements and when the bullets are fired.

TankControl- The TankControl class handles all the inputs to control the tanks when the keys are pressed registering and giving the output of moving the tanks.

TileManager- The TileManager class is where it takes the images of everything used in the game and puts where we design our maps to be, such as the tank, the power ups, and the walls rendering all the images together.

TRE- The TRE class is where everything the game runs all together. It contains the threads and game loop that runs the whole tank game. It also contains the movement system where tank players are able to move and shoot using keyboard keys. This class also instantiate lots of the rendering and logic behind the game.

Wall- The Wall class extends GameObject class. This class is one the game object that needs to interact with other different game objects.

9 Project Reflection

My thoughts on this term project, is that it was quite hefty in terms of work compared to the previous assignments. The difficulty came with the fact that most of the project design were based on what we came up with so it wasn't easy to know where to start exactly or how we should put this project together. I think once you got the core pieces of what a Tank Game comprised of, it was easier to have the design freedom of implementing the game how we would like. Although for me the designing aspect led me to stray away from some of the requirements and how I was putting the files together, the game turned out well and I was just glad to get things working properly into a functioning game.

10 Project Conclusion/Results

Overall this project was a good experience for learning Java programming, it took on many aspects of what we learned throughout our Summer session and assignments building upon each other. This is important for the future when we are required to build programs ourselves from almost scratch, taking what we know and trying to use some creativity to it for the designs. It was difficult not knowing where to start on the game, but taking care of the base work helped get things going and slowly putting each thing in the game until everything came together. There were many classes that we needed to put together for the Tank Game but trying to put what we learned and the resources from other classmates and online, helped push getting through the project. There were many errors I got stuck on and spent a lot of times trying to figure out what went wrong but some simple fixes solved the issues. Most of it was trial and error but as long as it eventually worked, that allowed me to keep on going. Even though this showed us how difficult it could be to program a game ourselves, it provided a great experience as what would happen in the real world when it comes to making video games from scratch using code.