CSC 413-02

Summer 2020

Tank Game Documentation

By: Dylan Shwan

ID: 917 262 260

https://github.com/csc413-02-SU2020/csc413-tankgame-dshwan

Table of Contents

Project Information

This section will serve as a general overview of the project while the introduction section will delve into some more technical details of the project. The term project describes within this report consists of the Tank Game. This report will detail my approach and the obstacles I faced.

Introduction

I implemented the Tank Game completely with Java. My main objective was to try and minimize coupling while increasing the level of cohesion within my code, especially since I failed to do so in previous assignments.

The tank game was large compared to the previous assignment, featuring various resources, features, and functionalities. This was initially very hard to tackle as I did not want to couple heavy dependency between my classes. My approach was a little unconventional in the sense that most things were implemented in a methodical fashion, but that my overarching goal was to get a working base product first.

Class diagrams were of a tremendous help to getting a scope of what had to be done. Despite me not completing my diagram on time, I urged myself to completing it, as it would be important to follow and not be intimidated by the size of this project.

Scope of Work

My focus within this project was to implement good structure that allowed ease of modification as well as potential future optimization. As this is my final semester at SFSU, I have thankfully picked up various guidelines and principles that allowed me to make good projects in this course and other courses.

Background / Game rule explanations

The section details the controls and the objective/rules of the game.

By clicking the provided .jar, you will be taken to the main menu in the following picture.

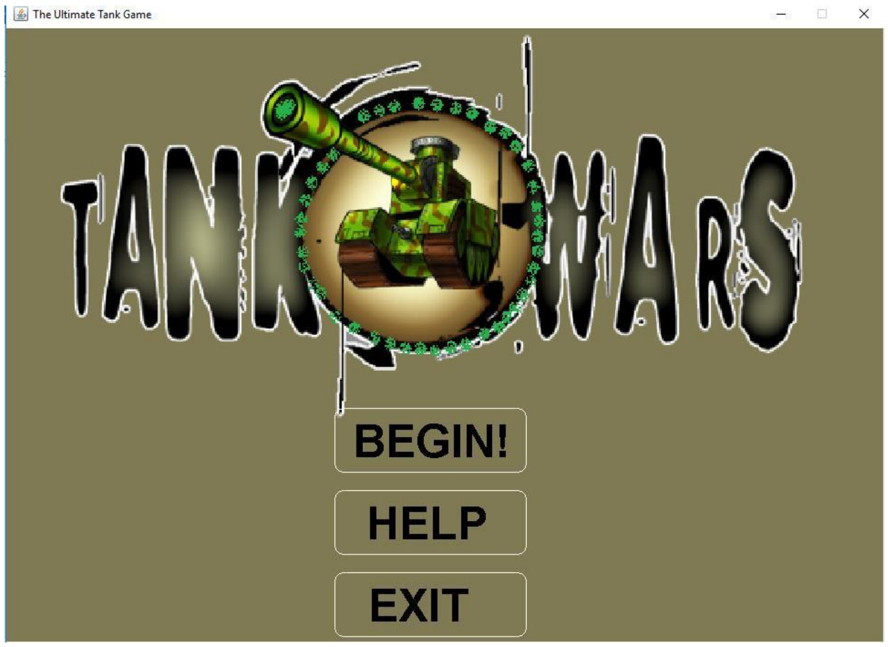
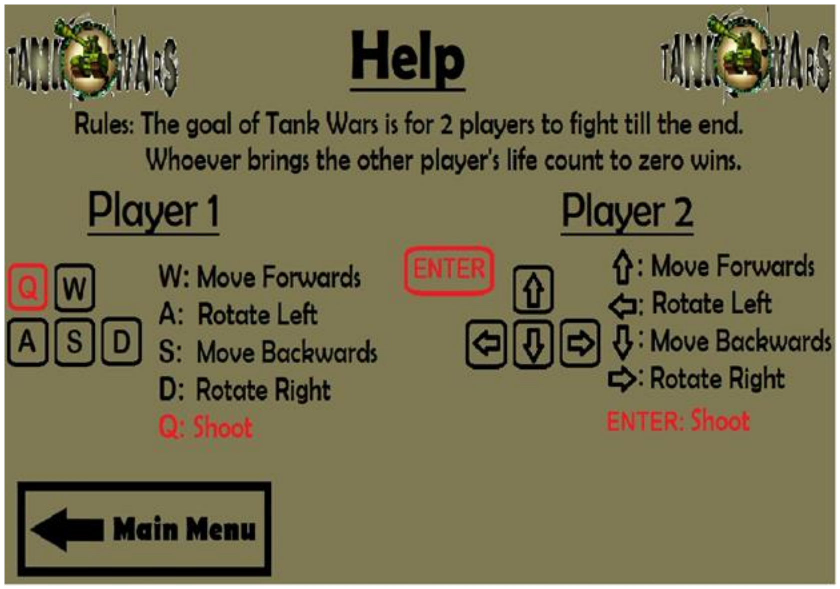


Figure (1): This figure shows the start/main menu screen of the Tank Wars game.

Players with their mouse can select the options **BEGIN!** to start playing the game, **HELP** to learn the controls of the game, or **EXIT** to leave the game. It is recommended to select **HELP** prior to starting a game and doing so pops up the following screen, detailing all the key assignments and objective. When selected, the following screen is shown.



Help

Rules: The goal of Tank Wars is for 2 players to fight till the end. Whoever brings the other player's life count to zero wins.

Player 1

W: Move Forwards
A: Rotate Left
S: Move Backwards
D: Rotate Right
Q: Shoot

Player 2

⇧: Move Forwards
⇦: Rotate Left
⇩: Move Backwards
⇨: Rotate Right
ENTER: Shoot

Figure (2): This figure shows the "Help" screen of the tank game.

In addition, Power Ups that increase a player's health or speed can be picked up to give players an advantage over their opponent. Once players are confident in their understanding of how the game works, they may select **MAIN MENU** on the bottom left to return to the menu screen.

Once the players return to the main menu, select **BEGIN!** to start a match. At the start, you will see the game laid out as such:

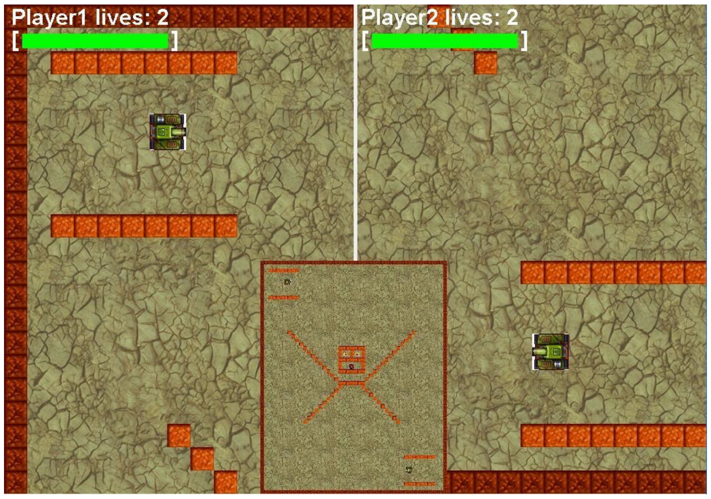


Figure (3): The default positioning and layout of the match once the player selects **BEGIN!**

There are two split screens with each screen belonging to a player. Each screen features a Health Bar, Lives Counter as well as a close-up of the player's tank. Additionally, there is a mini map provided in the center to witness the full scope of the map as well as player positions. The map was placed in the center to mitigate any potential advantage a player might have. The dark red walls surrounding the perimeter are the map's borders and cannot be destroyed. Conversely, the lighter colored walls within the map are breakable and are eliminated upon receiving enough damage (2 shots).

Once either player loses both their lives, they will be greeted by the following screen:

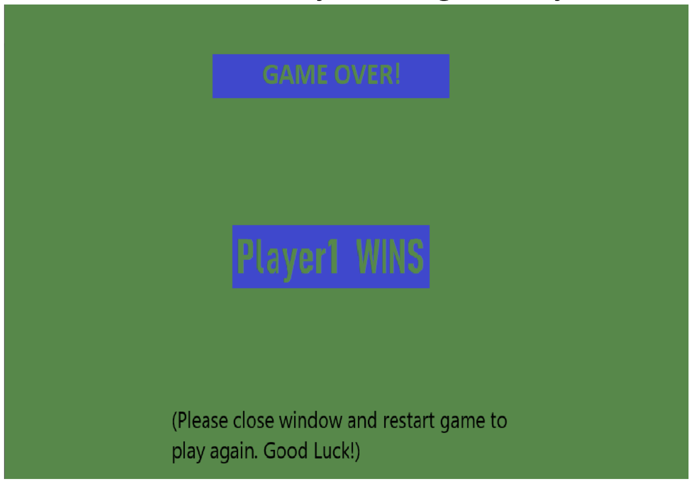


Figure (4): The victory screen indicating that Player1 has won after defeating Player2 by depleting their lives.

Development Environment

a) Version of Java used: Java

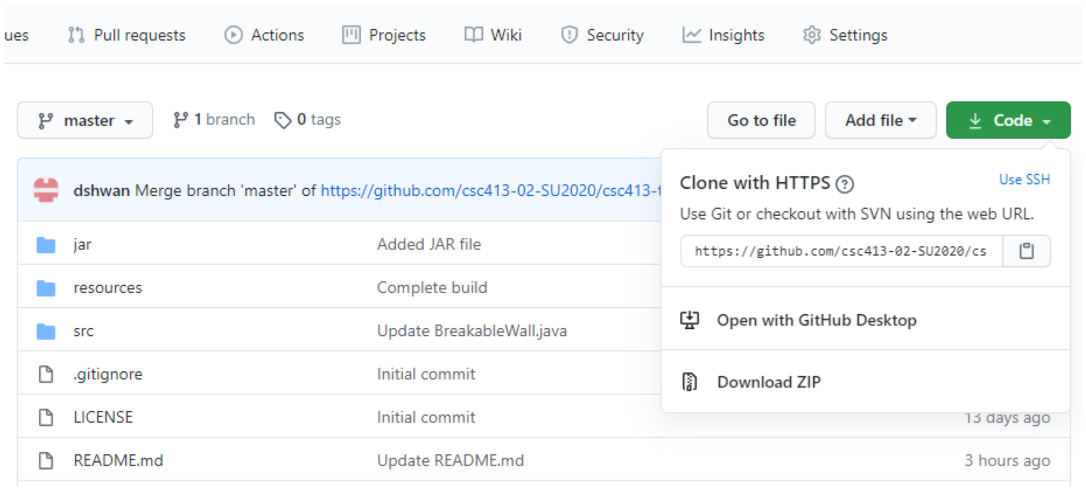
14.0.1 b) IDE used: IntelliJ Idea

20.1.1

c) Resources: Most of the images used in this project were provided by the instructor on iLearn. The only two external images were the health boost and speed boost powerups, which were derived from "Game-icons.net". As for code resources, I used and modified the Tank Rotation Example given via iLearn. All rights and licensing belong to their respective owners. All the code used in these two projects was purely my own apart from the base code given in the Tank Rotation Example.

How to Build and Import the Game

The easiest way to run the Tank Game is to double click the provided JAR file on a machine with the previously mentioned version (or later version) of Java. Alternatively, you may import and build the code within IntelliJ IDE.



dshwan Merge branch 'master' of https://github.com/csc413-02-SU2020/csc413-t

Use Git or checkout with SVN using the web URL.

https://github.com/csc413-02-SU2020/cs

jar	Added JAR file	
resources	Complete build	
src	Update BreakableWall.java	
.gitignore	Initial commit	
LICENSE	Initial commit	13 days ago
README.md	Update README.md	3 hours ago

Figure (5): Go to the repository link and click the green **CODE** button on the right. From there, select the option **Download ZIP** to download all the necessary content within a ZIP folder. Locate the ZIP folder from your downloads location and extract the ZIP folder to your preferred location.

Afterwards, launch IntelliJ and select **Import Project.** Then select wherever you extracted the ZIP file and press **OK**. From there, follow the following sequence:

- Select **Create Project from Existing Sources** and then hit **NEXT**
- Verify the **project name** matches the **project location** (should be "csc413-tankgame-dshwan") before hitting **NEXT**
- Proceed with hitting **NEXT** on the next few prompts until you reach **FINISH**, in which you would click on.

- Once the project is created, navigate to the **Project Structure** window (Ctrl+Alt+Shift+S) and verify that the “src” folder is selected as a source and the “resources” folder is selected as a resource folder.
- Click **Build** and select **Build Project**
- Right-Click the **GameWorld** class and select “Run ‘GameWorld.main()’”

a) Building jar:

I used IntelliJ’s Jar building feature to build my jar file. Through **Project Structure**, select Artifacts and create a new jar file with the rebuild each time option selected. This ensures that a new jar is generated within the jar folder each time.

b) Commands to Run the jar:

As mentioned earlier, the easiest way to run the jar file is by double clicking it on a machine that has the proper version of Java. Alternatively, you can run a terminal and type **java – jar [filename].jar** to run the jar via command line. I used Git Bash for testing.

Assumptions Made

My biggest assumption was that the tank controls would have to be very fluid and sensitive enough to where a player can navigate and avoid their opponent effectively. This included moving diagonally and rapid turning capabilities. It is implied that the player is only required to hold a key down rather than rapidly pressing a key for the sake of simplicity and overall better game feel.

Tank Game Class Diagram

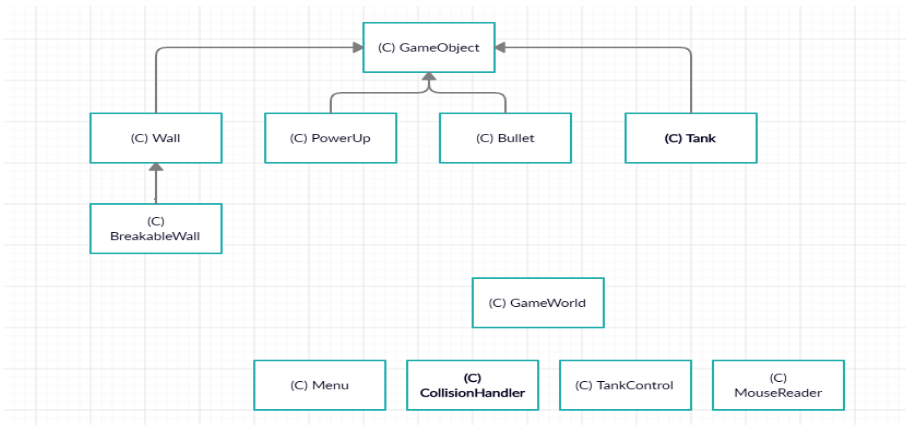


Figure (6): Tank Game class diagram.

Class Descriptions:

- **GameWorld**: This class is the entry point of our program and handles creation along with maintenance of the game. This class features a vast amount of variables, so I will primarily focus on the overall responsibility of the class and how it works instead of going over each variable in great detail. The init() method of GameWorld reads and sets the resource images used to provide animations and various displays within the game. The init() function is also used to create the map of the game which is stored as a single array. The init() method is also used to initialize all other map/game related items such as the player, player control and powerups. The drawImage() method is used to display the game as a whole. It handles centering the display on the player, health bars, number of lives and the appropriate results screen. All the functionality works alongside other classes using getters to receive the various states of the player.
- **GameObject**: This is an abstract class that holds x and y positions & velocities. It also holds a Rectangle, which helps with implementing proper collision. There are getters and setters for all the previously mentioned variables except the Rectangle, which only requires a getter since it is set within this class. The GameObject class includes 3 abstract methods utilized by every game object: update(), drawImage(Graphics2D), and collision(). With this, we are able to be efficient and reduce the amount of duplicate code within the program as well as promote simplicity such that the update() function can be called on a game object instead of having to call its own separate functions.
- **TankControl**: This class has private ints to hold each necessary button from the keyboard. The class implements the KeyListener interface, so it implements the methods within that interface (keyTyped, keyPressed, and keyReleased) which all take in a KeyEvent object. The class holds a data field of the type Tank. Appropriate methods for these data fields are called to set Booleans that they have been pressed or released.
- **CollisionHandler**: This class has a single method (HandleCollisions(ArrayList<GameObject>)) which takes in an ArrayList of GameObjects, handles the collisions between them, and returns a new ArrayList of GameObjects. The CollisionHandler class uses a nested for loop to get an object and compare it with all objects after it in the list. It uses the instanceof operator to determine the type of objects. Getters for the GameObject Rectangle are used to receive rectangles and compare them for intersections with the built in intersects() method of the Rectangle class. Appropriate handling is done for each different occurrence of a collision.
- **Wall class**: The wall class is a rather simple class that allows for the creation of Wall objects. It is very important to note that this class extends the abstract GameObject class and thus implements all its abstract methods. The Wall class has a boolean variables used to denote which image should be used when the drawImage function is called. In the Tank Game, there is a single boolean variable to denote if an instance of the Wall class represents a background image or a regular wall. Since the wall class extends GameObject, each Wall has an x and y position which is initialized via the Wall constructor.

- **Tank**: Easily the most crucial extension of the GameObject class since it adds a lot of functionality to it. We use the proper boolean variables which were set in TankControl to move the tank in the update method. A SpawnBullet() method exists to spawn bullets with a given x, y, vx, vy, and angle. For the sake of realism, there is a restriction on how often a player can call this method due to the LastFired variable which holds the last time a bullet was fired. Players can only fire once per second since real life tanks do not spit rapid fire. The class has methods for rotation and for moving in various directions. A boolean variable is used to check if a tank is currently speed boosted. If speed boosted, it travels at roughly four times the original speed. We also have a time variable to control the speed boost and to ensure that it is shut off after one second.
- **Bullet**: A class that extends the GameObject, though not to the degree of Tank. A string is used to denote the owner of the bullet while an isInActive boolean variable is used to mark whether the bullet is inactive. Inactive bullets are removed from the game_objects ArrayList in the main method of GameWorld. Three images are stored within this class: bullet_img, big_explosion_img, and explosion_img. The bullet_img holds an image of the bullet. The explosion_img is used to display the image of an explosion when a Breakable Wall has been hit. The big_explosion image is used to display the image of a more substantial explosion whenever a tank shoots another tank. Bullets are marked inactive if they cross the boundaries of the game map to prevent holding unnecessary Bullet objects. The Bullet is marked as inactive a couple instances after colliding with an opposing tank to provide enough time for the explosion to show.
- **PowerUp**: This class extends GameObject since it has a x and y location as well as an image and a Rectangle. I used isHealthBoost and isSpeedBoost to allow for control over which image should be shown and how collisions can be handled. In the collision method, we simply set a boolean variable isActive to false. This allows us to later remove the used up PowerUp object.
- **BreakableWall**: This class adds some slight functionality to the Wall but is different enough to warrant a separate class. It differs from the Wall class because it has a health field as well as some methods to manage its health. A boolean variable is used to mark it as “dead” or “alive”, so it can be cleaned up in main and removed from the game_objects array when appropriate.
- **Menu**: This class was included since a typical game tends to have a main menu rather than dropping you into the game right away. Not to mention it provides an opportunity to provide details of the controls without having to refer to an external file. This class displays curved rectangles used as buttons on the main menu as well as the text used for options in the main menu.
- **MouseReader**: This class implements the MouseListener interface, so it must also implement all the methods specified within the MouseListener interface. The only parameter I included was whether the mouse was pressed, so mousePressed(MouseEvent) was the only required method that wasn’t left blank. In this method, I used a static game_state variable from GameWorld which decides where to register clicks. The game_state variable returns to the proper game_state when buttons are pressed with the mouse.

Self-Reflection

This term project was fairly difficult, but far more engaging and entertaining in comparison to previous assignments. The areas where I had the most difficulty were making the controls of the tank smooth, but somewhat realistic as well as getting the collisions to work properly. After looking at the examples and lectures on iLearn, I felt comfortable with these concepts. Apart from large websites, this is the biggest project I have ever tackled, especially one with this amount of interactable methods and classes. I approached the game in a piece by piece aspect where I tried to get something minimal working before adding to it, solving bugs, and organizing my code properly. It is a far more leisure and practical approach compared to what I did on previous assignments, where I did everything in one go and had to scramble to find issues.

Conclusion

In this project, I designed and implemented a Two-Player Tank Game. The game was completely implemented in Java where proper principles and guidelines were adhered to provide a nice and effective coding experience. I am very grateful for having the opportunity to tackle this project and have learned a lot more about Java and overall better coding practices than ever before. I'm thankful for the work and passion my instructor has for this subject.