CSC 413 Term Project - Tank Game
(Wizard War for my theme take of the game)
Summer 2024

Bryan Lee
922649673
Section 1-R4

https://github.com/csc413-SFSU-SU2024/csc413-tankgame-BryanL43

Introduction

Project Overview:
For this project, I created a themed version of the base tank game, turning it into more of a wizard duel with spells as projectiles. The primary focus of this project, deviating from the tank game, was implementing a variety of spells, each with its unique attributes and abilities. The game has objects interact with each other through collision, invoking actions such as damage, healing, animations, etc.

Introduction of the Tank game:
In my design of the game, I used a wizard one-on-one duel, where both players play on the same device on a split screen. The player will try to bring the opposing player's health to zero, thus losing a life, with four different spells to choose from: fireball, magic bullet (bouncing projectile), zap, and wind blade. There are three lives per player, indicated by the cropped images of the player's character on the bottom UI. There are also power-ups, giving either instant health, slow regeneration, shield, or faster casting speed, giving the game a unique twist.

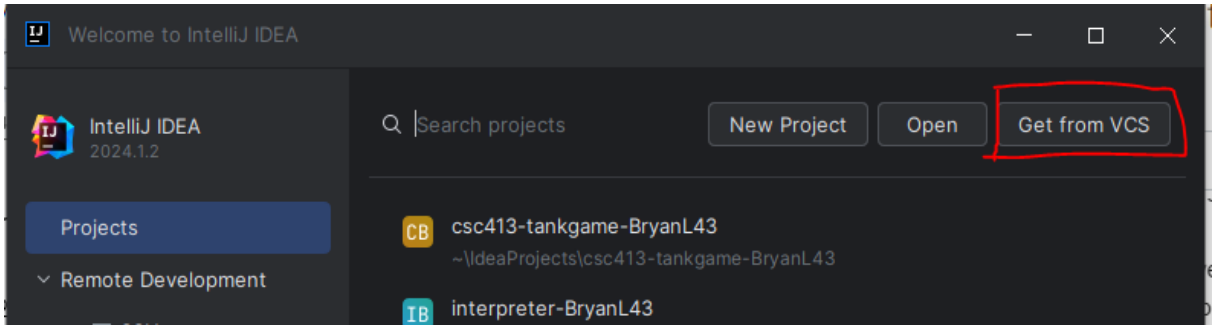
Development Environment

a) The version of Java used: Oracle OpenJDK 21.0.2
b) IDE used: IntelliJ
c) No special libraries or resources are used aside from assets (images & gifs) found on the Internet, primarily Google Images, which I have already implemented into the project.

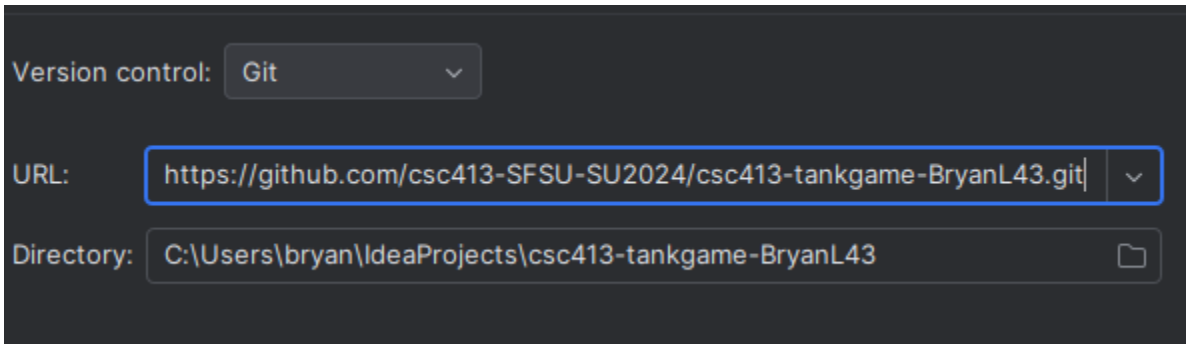
How to build the project

Building the project in IntelliJ:

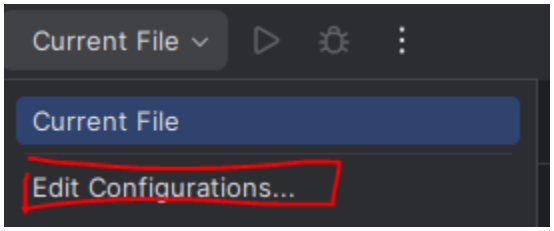
Step 1: Click “Get from VCS”



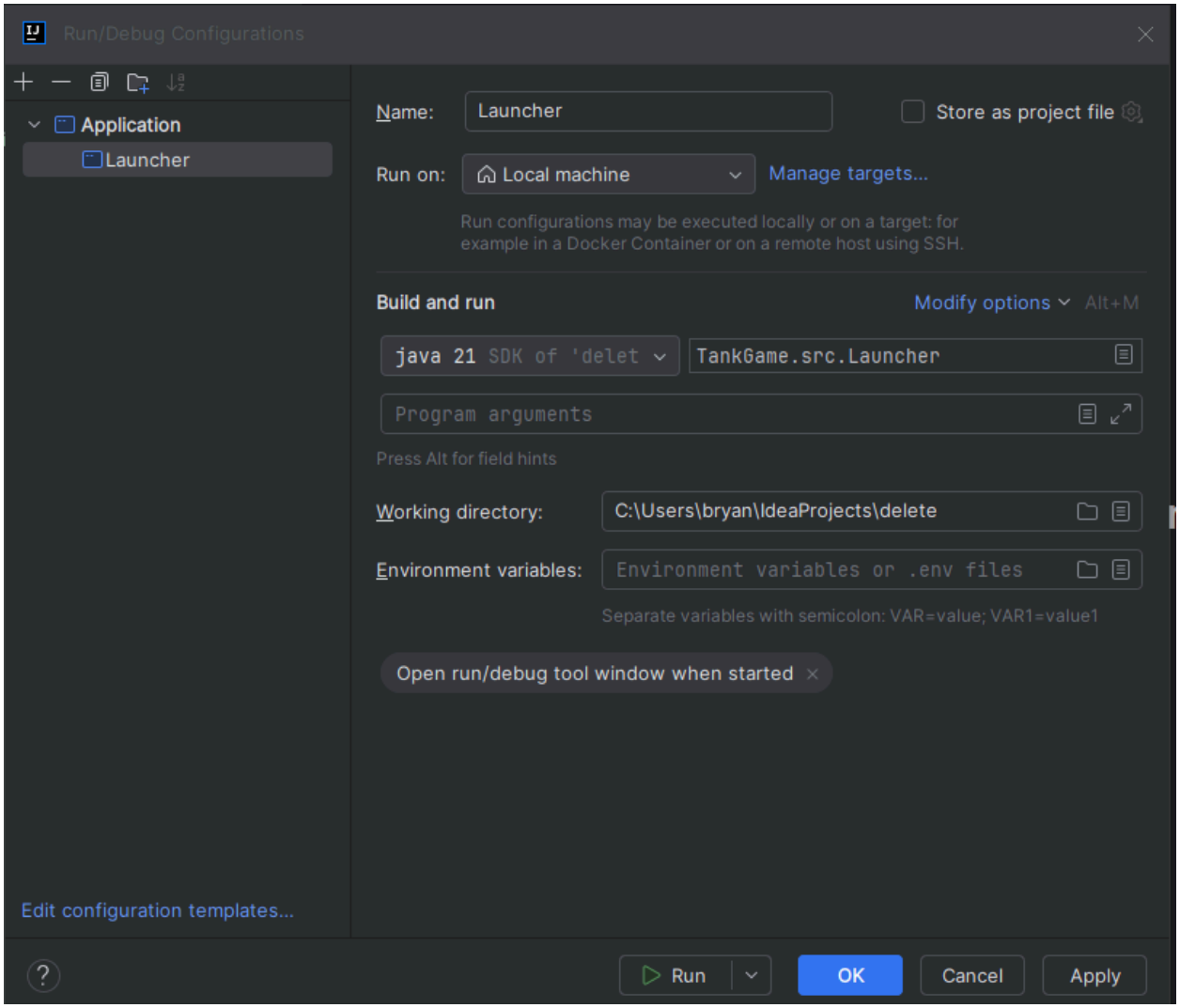
Step 2: Paste the GitHub “git” link into the link and click clone



Step 3: Click the following “Edit configuration.”



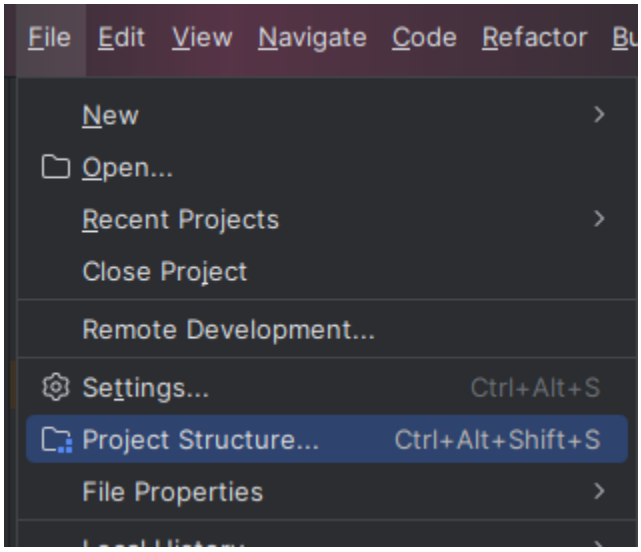
Step 4: Create an application run configuration as follows.



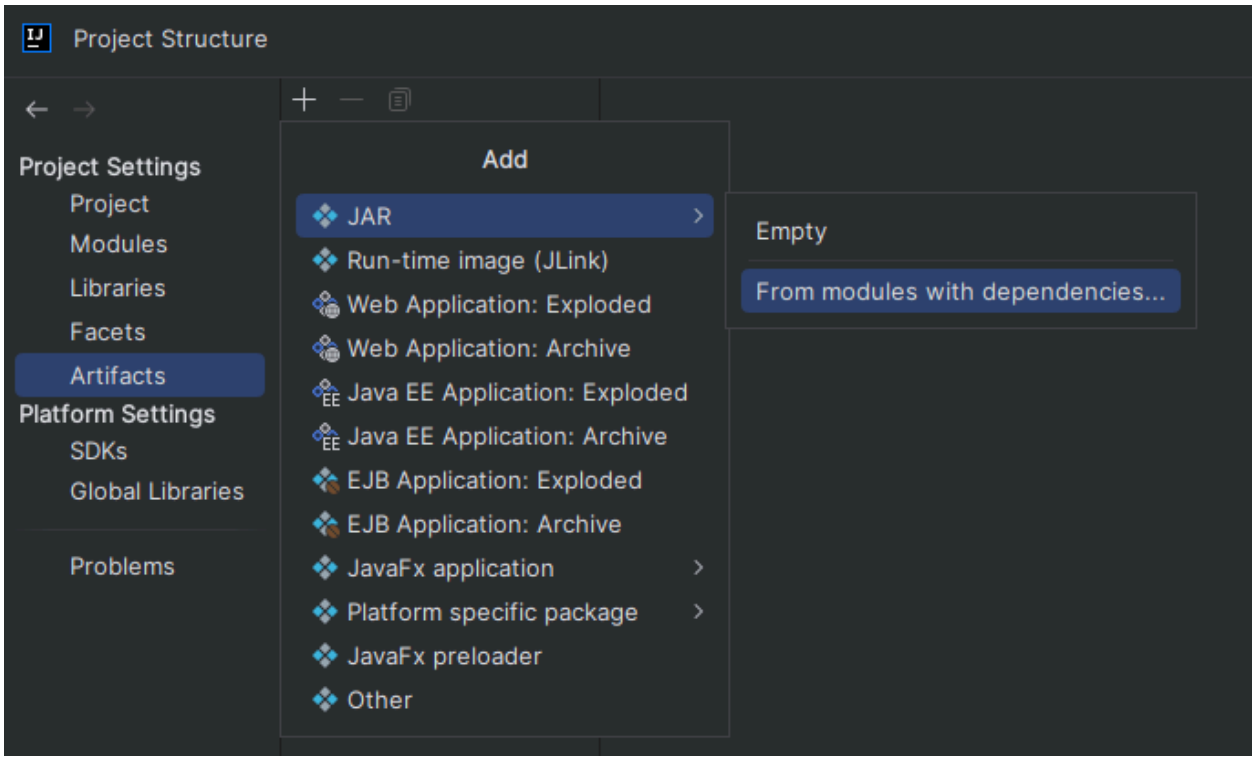
Step 5: Click “Apply” and then “Run” to build and run the game.

Building the JAR in IntelliJ:

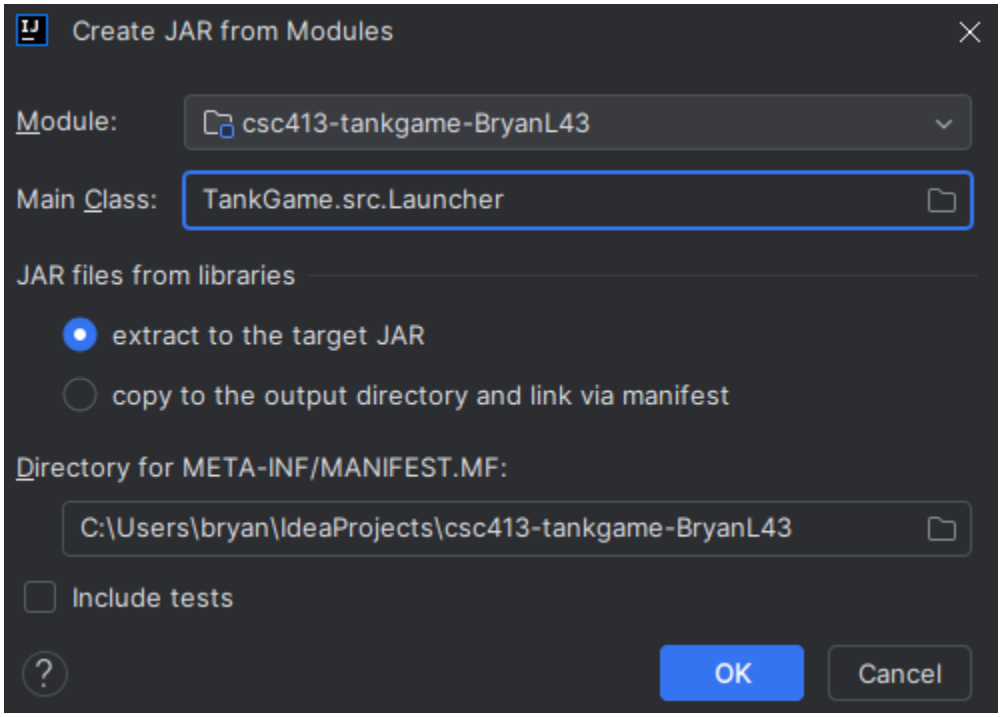
Step 1: Open up “project structure”



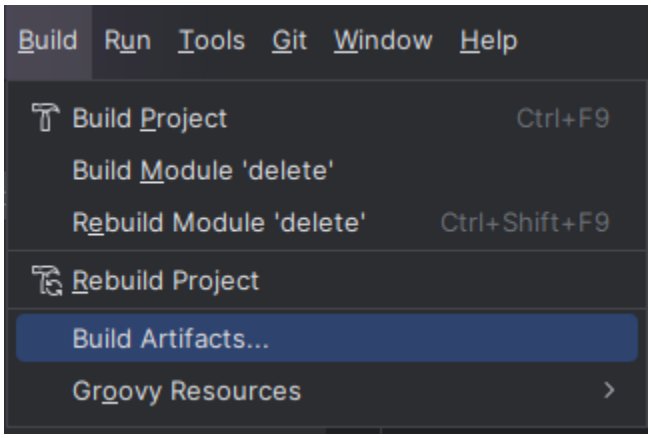
Step 2: In the “Artifacts” tab, select the following



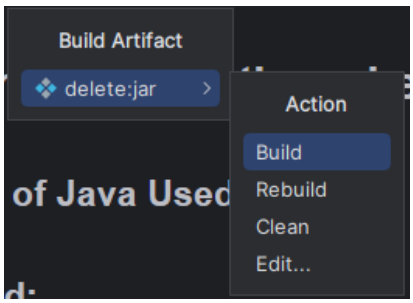
Step 3: Input the appropriate “Main Class” to be directed to the launcher and click “Ok.”



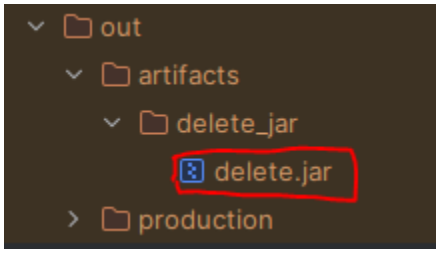
Step 4: Under the “build” tab, select “Build Artifacts.”



Step 5: Simply select the jar and “Build.”



Step 6: Locate the JAR in the out folder.



The command needed to run the built jar:

Ideally, running the built jar will only be required if you have directly downloaded the zip from GitHub. First, ensure you extract the zip, and then in either PowerShell or the command prompt, change the directory to the extracted folder.

Ensure the path looks like this when “ls” is called:

```
PS C:\Users\bryan\Downloads\csc413-tankgame-BryanL43-main> cd csc413-tankgame-BryanL43-main
PS C:\Users\bryan\Downloads\csc413-tankgame-BryanL43-main\csc413-tankgame-BryanL43-main> ls

    Directory: C:\Users\bryan\Downloads\csc413-tankgame-BryanL43-main\csc413-tankgame-BryanL43-main

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d----           8/4/2024  5:55 PM                jar
d----           8/4/2024  5:55 PM                META-INF
d----           8/4/2024  5:55 PM                TankGame
-----           8/4/2024  5:55 PM            138 .gitignore
-----           8/4/2024  5:55 PM            419 csc413-tankgame-BryanL43.iml
-----           8/4/2024  5:55 PM           1074 LICENSE
-----           8/4/2024  5:55 PM            976 README.md

PS C:\Users\bryan\Downloads\csc413-tankgame-BryanL43-main\csc413-tankgame-BryanL43-main>
```

Finally, you can simply execute the following command to run the built JAR:

```
java -jar .\jar\csc413-tankgame-BryanL43.jar
```

How to run my game:

To run the game, you could either build the project with IntelliJ or execute it with the built JAR, as explained above. Next, is to press “Start” and play the game simply:

How to run the Game

Rules: N/A, as all restrictions have already been implemented within the game.

Controls:

	Player 1	**Player 2**
Forward	W	U
Backward	S	J
Rotate left	A	H
Rotate right	D	K
Shoot (Hold & Release when finished casting)	F	L
Previous Spell	Q	Y

Next Spell	E	I
Recharge spells (dismiss recharge included)	R	O

Note: To shoot, you must charge the yellow bar by holding the “Shoot” button. To recharge spells, you must wait three seconds upon clicking the button or dismiss with the same button.

Assumption Made When Designing/Implementing the Project

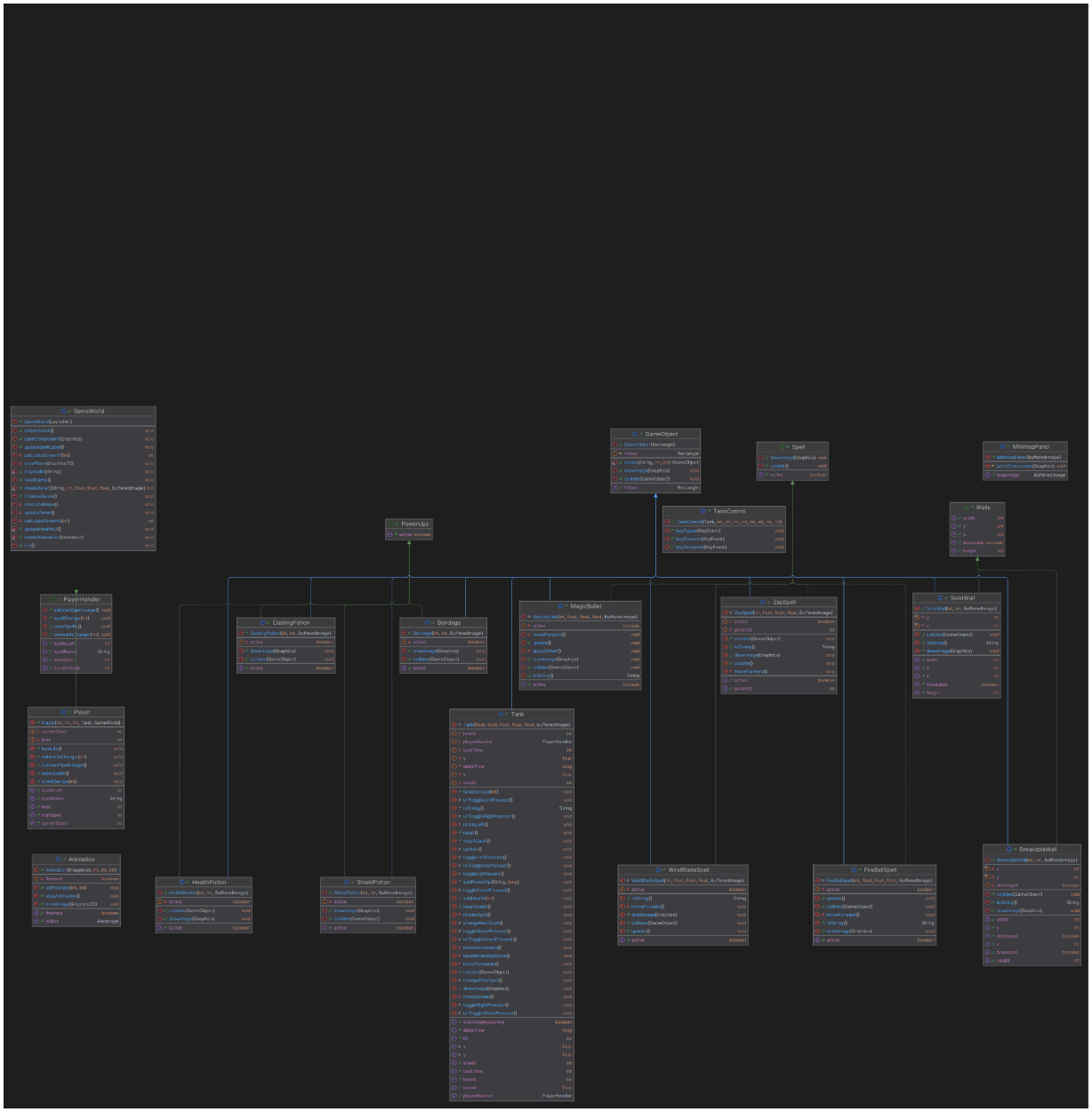
One assumption I made when designing the project was that the “GameWorld” object handles everything that happens in the game, especially the map, projectiles, animations, UIs, and sounds. Handling everything in the “GameWorld” did induce some concurrency issues related to removing animations, sounds, and objects on different threads. However, that was quickly resolved with the “removeIf” function when clearing out a list. Handling most of the functionality in the “GameWorld” also required me to make some static methods to enable pass-by-references, ensuring that I do not pass the entire “GameWorld” into other objects. Another assumption I made was my strict enforcement of encapsulation, where objects will have to access another object’s data through mutators, accessors, and custom methods. Doing so limited the time I needed to pass entire objects into another object, only to exceptional cases, like the player class, and reduced accessing/modifying data incorrectly.

Implementation Discussion

On the other hand, I did have to make a few implementation designs I did not like, but it was the most efficient way. The first one is the sound implementation, where the primary issue was that the audio was playing under different threads. I decided to delete the Audio object and create a new one every time I needed the audio. However, that was not possible with the Java package I was using, while the other choice was to install JavaFx packages, straying away from the base design of the game. The simple solution

was to stop the audio and start playing it again when I invoked it before the previous call had finished. Doing so obviously caused issues with sounds cutting off midway through playing. Another design choice I made was implementing animations as GIFs to make things more simple. Since I deviated from the base tank game, I needed a lot more animations, and the easiest way to implement them was not the frame-by-frame rendering but by loading a GIF. I also followed OOP as much as possible, creating abstract classes to group objects, such as “PowerUps,” “Spells,” and “Walls.” I utilized the interfaces to enforce necessary methods in the inherited class and made it easier to reference the broader abstraction when needed.

Class Diagram



A better quality and more interactable image is located here:
https://github.com/csc413-SFSU-SU2024/csc413-tankgame-BryanL43/blob/main/TankGameDiagram.png

Class Descriptions

Animation:
The animation class creates an animation object based on an imported gif. It provides information such as when the animation finishes playing, hitbox, location, and ability to stop the animation.

Bandage:
A powerup object that instantly heals a player for 15 health.

BreakableWall:
As the name implies, it is a breakable wall object.

CastingPotion:
A power-up object that allows a player to cast spells 25% faster.

FireBallSpell:
A fireball spells projectile that has a larger hitbox, and its explosion can damage a player if caught nearby.

GameObject:
This is an abstract class representing every interactable object in the game. It handles their types (walls, potions, etc.) and enforces signature methods for hitbox management, image rendering, and collision detection.

GameWorld:
This object houses everything in the game. It renders objects, maps, animations, UIs, and split screens. It hosts the main game loop and checks for collision and interactions such as collision.

HealthPotion:
A power-up object that heals a player for 30 health points over 15 seconds, similar to the bandage.

MagicBullet:

A spell projectile that bounces twice on collisions or deals ten damage if it hits the opposing player. It fires in a burst of five shots per spell cast.

MinimapPanel:
This object simply handles rendering the minimap object.

Player:
This object takes on the observer pattern, handling any updates in the player's health and power-ups. It tracks the lives each player has remaining, controls the number of spells remaining, and handles spell changing.

PlayerHandler:
Interface for the Player class.

PowerUps:
Interface for all the power-up objects.

ShieldPotion:
As the name implies, this object gives the player 20 shield points and expires in 20 seconds if not destroyed.

SolidWall:
An unbreakable wall.

Spell:
Interface for all the spell projectiles.

Tank:
The player of the game. It hosts the player's movements. The tank object also handles collisions, power-up abilities, and spell-casting and creates the spell projectiles. Although it is named "Tank," as I built the wizard game on top of the base tank game, the only difference is the wizard asset used to portray the player.

TankControl:
Tank control that handles the keypress actions of the player.

Walls:
Interface for the solid and breakable walls.

WindBladeSpell:
A spell projectile that has a high velocity that can cut through breakable walls and spells like butter.

ZapSpell:
A lightning-based spell projectile with a unique collision check, it explodes with an enormous zap effect within a 60-radius of an opposing player, dealing massive damage. This is by far one of the more complex projectiles in the game.

EndGamePanel:
The end game panel offers to restart or exit the game.

PlayerOneWonPanel:
Displays a UI that says, “Player 1 won.”

PlayerTwoWonPanel:
Displays a UI that says, “Player 2 won.”

StartMenuPanel:
The start game panel offers to start or exit the game.

Audio:
The audio object handler. It creates audio and allows it to play, loop, and stop.

Pair:
A generic pair object that handles spells left and powerups.

ResourceManager:
Acquire and load the appropriate asset from the “resources” directory.

GameConstants:

Set of game constants like UI size, game world size, and game screen size.

Launcher:
Starts the game.

Self-reflection on Development

I found this assignment fun, especially with my custom implementation of the game. Although I had to put in a bit more effort, the game didn't deviate too much from the base tank game besides the unique projectile system. The assignment absolutely solidified my knowledge of abstraction and inheritance, which I lacked until now. To be more specific, it helped me realize the importance and usefulness of OOP. Some other things I got to tinker around with were generics and design patterns. Although I learned them in previous courses, I've only understood the theory and never put them into practice. I was able to use a generic pair object over a dictionary (not to have to use multiple data structures) and the observer design pattern for the player object. I did have one thing I was frustrated about, particularly the multi-threading aspect. It deviated a lot from what I have learned in Operating systems, and with a provided base game, there wasn't much leeway in changing the program. I also worked on the project fairly quickly, making most of the lectures not so helpful to my game. If I had more time and had the knowledge ahead of time besides working ahead of the slow-paced lecture, I would have been able to handle the gameplay under one thread rather than working around multi-threading and facing concurrency issues.

Project Conclusion and Results

I am satisfied with how my project turned out. Although I feel like I've used some underhanded tricks to cover up the threading issues, I have resolved and minimized a majority of the errors. I also used some prior knowledge of game development to create my wizard game, especially the aspect where there is a "game" object that hosts everything else visible to the player. However, I do believe that I could have improved the audio a bit more if I had more time. I would have used a more versatile audio library

that would enable the audio to be destroyed and not played on one thread. Overall, this project really solidified my OOP fundamentals and allowed me to put the theories I have learned into practice.