

# *CSC 413 Term Project Documentation*

## *Fall 2024*

*Michael Thompson*

*CSC 413.01*

[https://github.com/michaelt-04/IsaacLit](https://github.com/michaelt-04/IsaacLite)

[e](#)

[Table of Contents](#)

<b>1 Introduction.....</b>	<b>3</b>
1.1 Project Overview.....	3
1.2 Introduction to My Game.....	3
<b>2 Development Environment.....</b>	<b>4</b>
2.1 Version of Java Used.....	4
2.2 IDE Used.....	4
2.3 Special Resources Used.....	4
<b>3 How to Build/Import My Game.....</b>	<b>5</b>
3.1 Build or Importing My Game in IntelliJ.....	5
3.2 How to Build the JAR.....	8
3.3 How to Run the JAR.....	9
3.4 How to run my game in IntelliJ.....	12
<b>4 About My Game.....</b>	<b>12</b>
4.1 Controls and Rules.....	12
4.2 Assumptions Made When Designing and Implementing.....	16
4.3 Class Diagram and Description.....	16
<b>5 Self-reflection on the Development Process.....</b>	<b>20</b>
<b>6 Project Conclusion.....</b>	<b>21</b>

# 1 Introduction

## 1.1 Project Overview

- The focus of this project was to create a game. We were asked to either make a Tank Wars Game in Java with two players going against each other with a split screen, or a game of our choice. The main goal of this project is to practice good object-oriented programming and create a game with good structure and organization. Instead of making the Tank Wars Game, I decided to try to remake “The Binding of Isaac”. The professor told us that if we wanted to make our own game, we would need to make our requirements for our game.

These are the requirements for my game:

1. The game must have a start screen.
2. The game must have an end screen showing that the player has lost.
3. The end screen must allow the user to restart or close the game.
4. The game is single-player against bot enemies.
5. The game must have a character that moves.
6. The game must have multiple rooms for the player to explore.
7. The game must have a health bar.
8. The game must have 3 or more power-ups (some of the power-ups from The Binding of Isaac)
9. The game must have unbreakable walls.
10. The game must have breakable walls.
11. The game must have 3 or more room layouts.
12. The game must have a character that can shoot projectiles that collide with walls.
13. The game must have a character that can shoot projectiles that collide with enemies.
14. The game must have 3 or more different animations.
15. The game must have 3 or more different sounds that play.

## 1.2 Introduction to My Game

- For my game, I decided to try to recreate “The Binding of Isaac”, which is a roguelike game that features semi-randomly generated dungeons, with many different items to pick up and enemies to fight. I knew that I most likely did not have the time to include all of the 716 items, 500+ enemies, or levels in The Binding of Isaac, so I wanted to make my version like a “lite” edition of the original game, which would have similar fundamentals but with a few of the enemies, items, and one level from the original game. This version would need to include a similar way of procedurally generating the map, as well as having similar gameplay and flow to the original. This was my main goal when

creating my game, I wanted to make sure that it felt like and had the same spirit as the original game.

## 2 Development Environment

### 2.1 Version of Java Used

- I used version 21 of Java to make my game.

### 2.2 IDE Used

- I developed my game in IntelliJ

### 2.3 Special Resources Used

- I used LibGDX to help develop my game, this resource was given to us on the main Canvas page of the project. LibGDX is a Java game development framework that offers many different tools and libraries to help with creating games.

I got it from this website: <https://libgdx.com/>

You can start a project using LibGDX by clicking “Get started” on the main page, then clicking “Setup a Project”, then clicking “Generate a Project”, and downloading the.gdx-liftoff.jar file. When you run this jar file, it will allow you to set up a new project with LibGDX and use it for your own game.

LibGDX allows for the use of Gradle as a build tool, I chose to use Gradle to make it easier to build my project. Gradle allows for multi-platform building using its build script, this allows anyone on any kind of device to build my project using Gradle, that way they don't need to do anything special or jump through any hoops. Gradle will also handle all of the dependencies needed for LibGDX automatically, which saves a lot of troubleshooting.

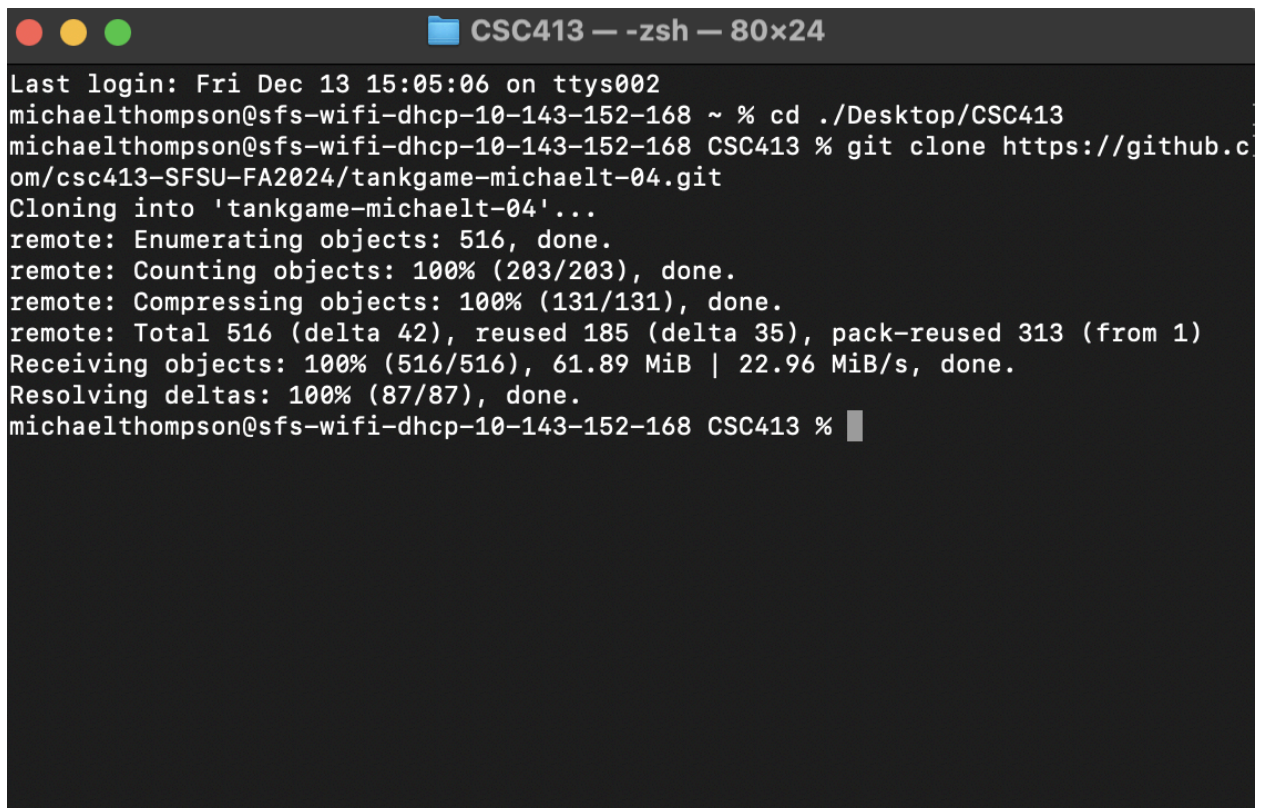
## 3 How to Build/Import My Game

### 3.1 Build or Importing My Game in IntelliJ

- To import my game into IntelliJ, you first need to clone my repository over to your computer by getting the clone link from GitHub and running the command

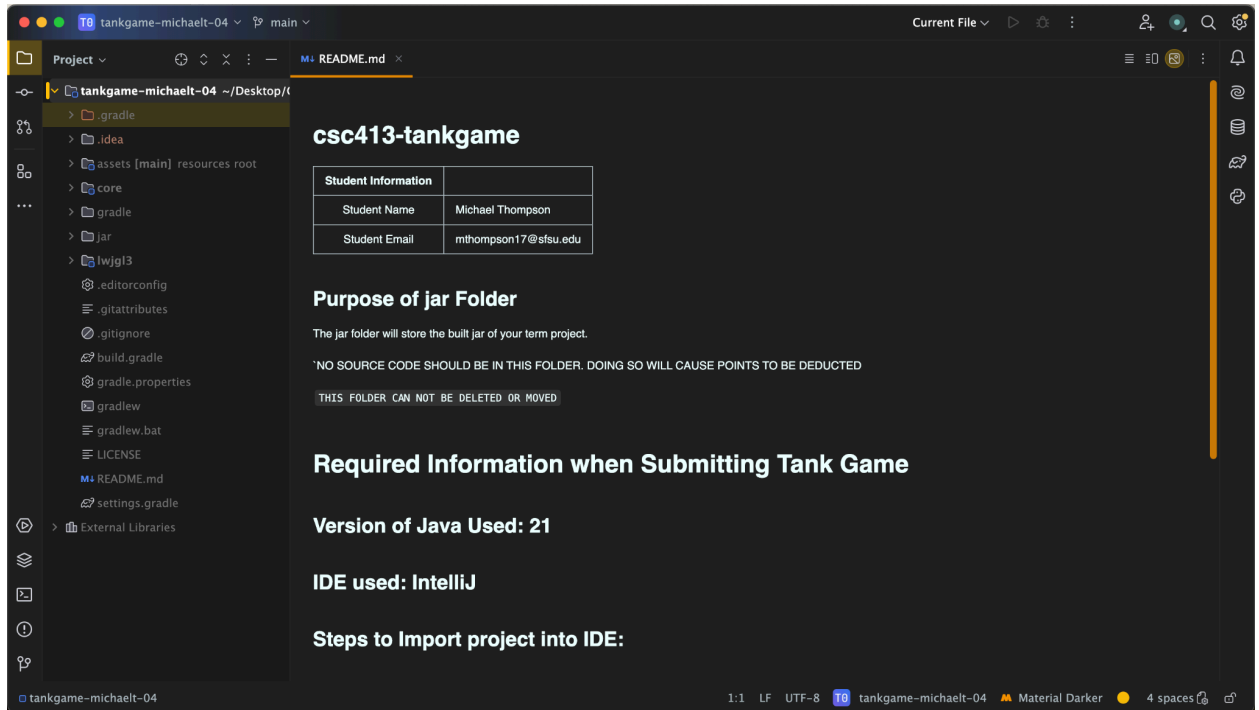
`git clone [link to repo]`

in the terminal in your desired directory.

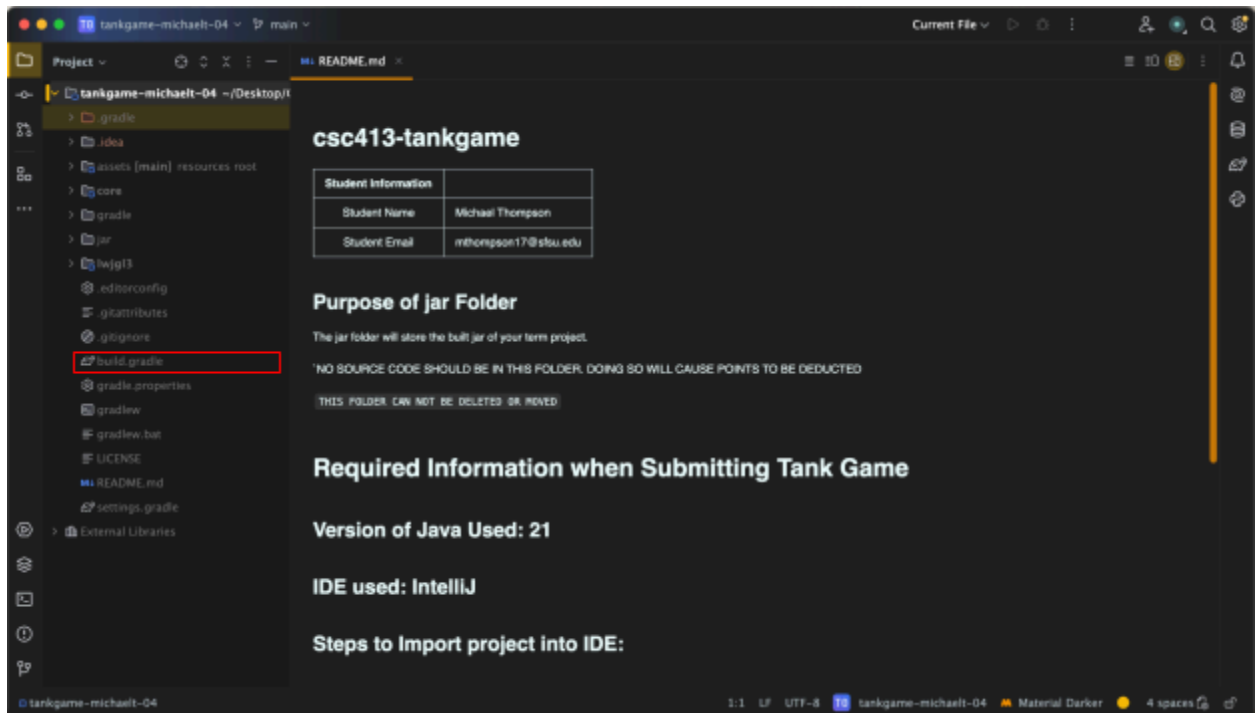
A screenshot of a terminal window titled "CSC413 — -zsh — 80x24". The terminal shows the following text: "Last login: Fri Dec 13 15:05:06 on ttys002", "michaelthompson@sfs-wifi-dhcp-10-143-152-168 ~ % cd ./Desktop/CSC413", "michaelthompson@sfs-wifi-dhcp-10-143-152-168 CSC413 % git clone https://github.com/csc413-SFSU-FA2024/tankgame-michaelt-04.git", "Cloning into 'tankgame-michaelt-04'...", "remote: Enumerating objects: 516, done.", "remote: Counting objects: 100% (203/203), done.", "remote: Compressing objects: 100% (131/131), done.", "remote: Total 516 (delta 42), reused 185 (delta 35), pack-reused 313 (from 1)", "Receiving objects: 100% (516/516), 61.89 MiB | 22.96 MiB/s, done.", "Resolving deltas: 100% (87/87), done.", "michaelthompson@sfs-wifi-dhcp-10-143-152-168 CSC413 %".

```
Last login: Fri Dec 13 15:05:06 on ttys002
michaelthompson@sfs-wifi-dhcp-10-143-152-168 ~ % cd ./Desktop/CSC413
michaelthompson@sfs-wifi-dhcp-10-143-152-168 CSC413 % git clone https://github.com/csc413-SFSU-FA2024/tankgame-michaelt-04.git
Cloning into 'tankgame-michaelt-04'...
remote: Enumerating objects: 516, done.
remote: Counting objects: 100% (203/203), done.
remote: Compressing objects: 100% (131/131), done.
remote: Total 516 (delta 42), reused 185 (delta 35), pack-reused 313 (from 1)
Receiving objects: 100% (516/516), 61.89 MiB | 22.96 MiB/s, done.
Resolving deltas: 100% (87/87), done.
michaelthompson@sfs-wifi-dhcp-10-143-152-168 CSC413 %
```

Once you have the repository on your computer, you can then import it as a project in IntelliJ. To do this you can open another random project, press “Ctrl+Shift+A” on Windows or “Cmd+Shift+A” on MacOS, then search “project from existing source” and click “Import Project from Existing Sources...”. Then you can click the repository you cloned, import it with Gradle, and you should see the project in your IntelliJ.

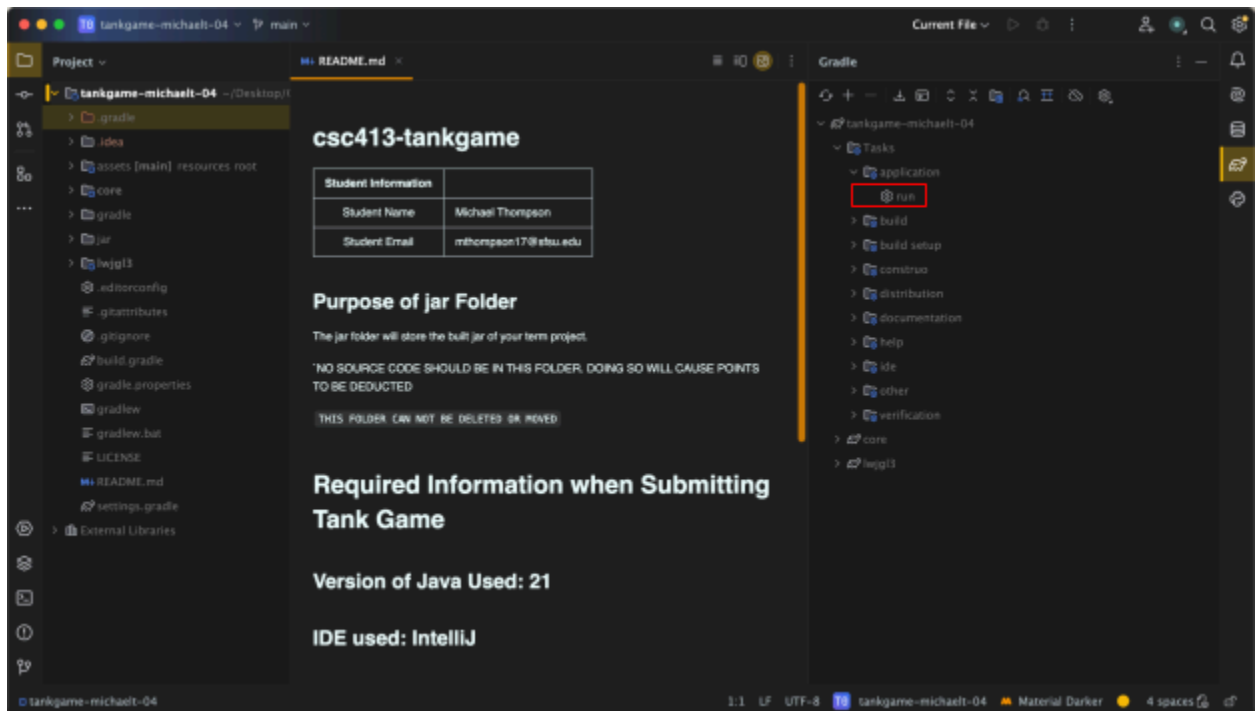


- In order to run my game in IntelliJ, you first need to build it by running the build.gradle script, which is in the root folder for the repository.

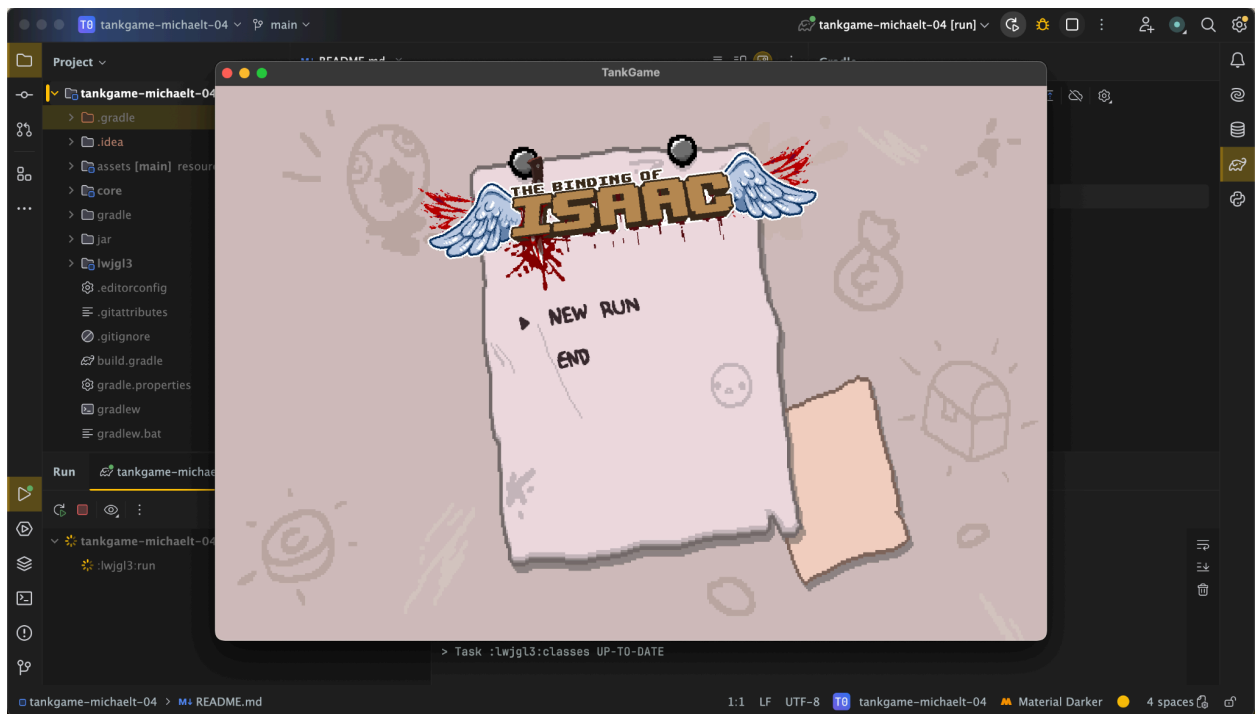


Once it is built, you should see a little Gradle logo on the far right of IntelliJ. Pressing this

Gradle logo, you can click “Tasks”, then “application” and you'll find the “run” task which will run my game in IntelliJ.

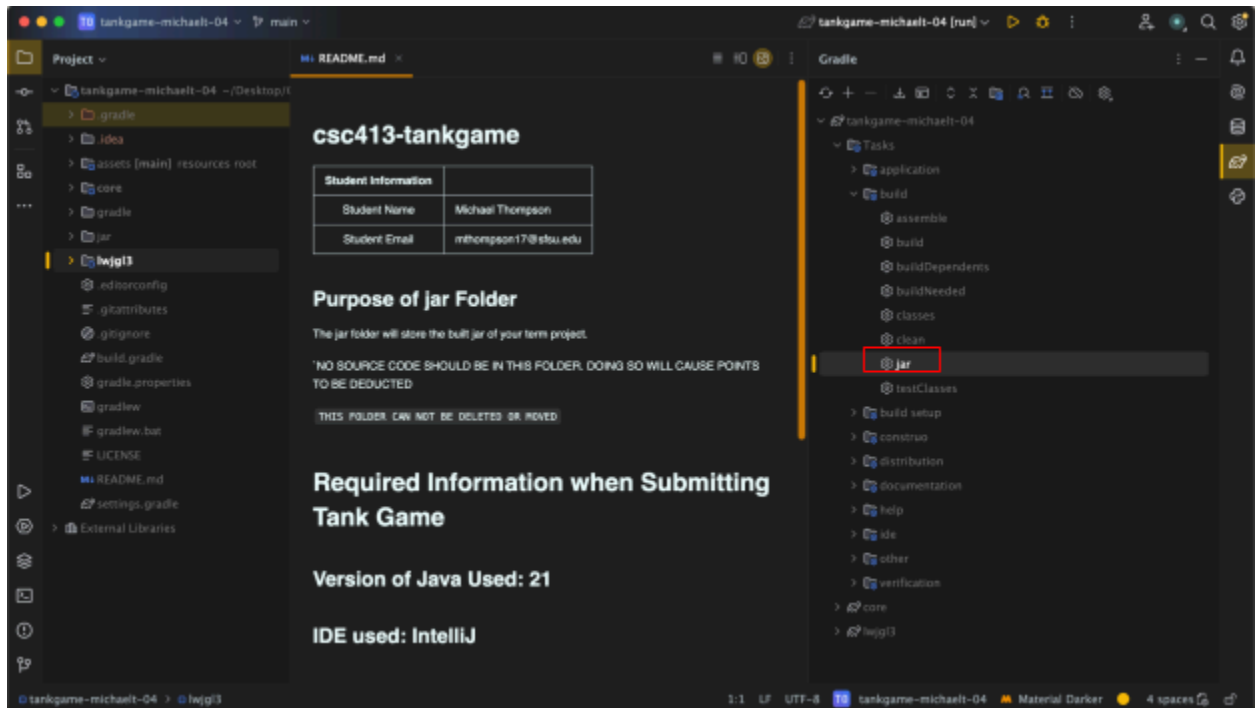


When you press run, it should launch the game and you'll see the start screen.



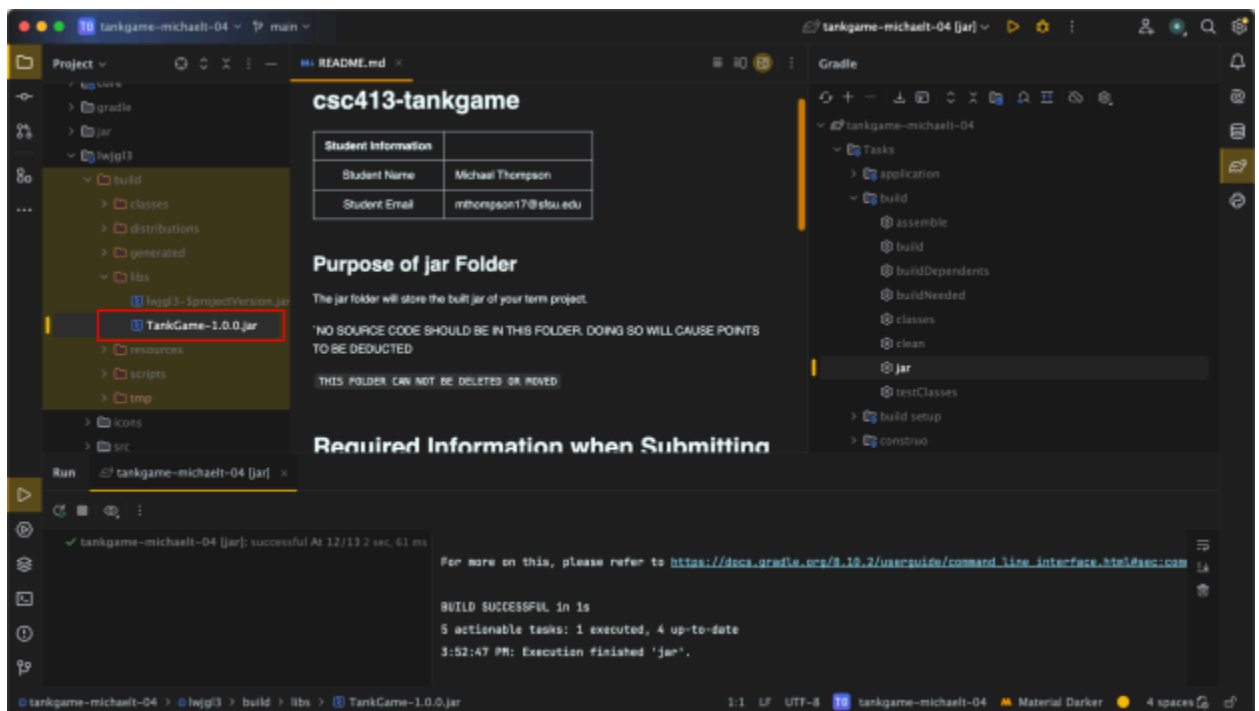
## 3.2 How to Build the JAR

- Using Gradle made it very easy for me to build the JAR file. By default, Gradle includes a “jar” task which will automatically make a JAR file for the project. This made it very easy to make a JAR file for the game. The “jar” task is under the “build” folder in the Gradle tasks.



After running the “jar” task, you can find the JAR file in lwjgl3/build/libs



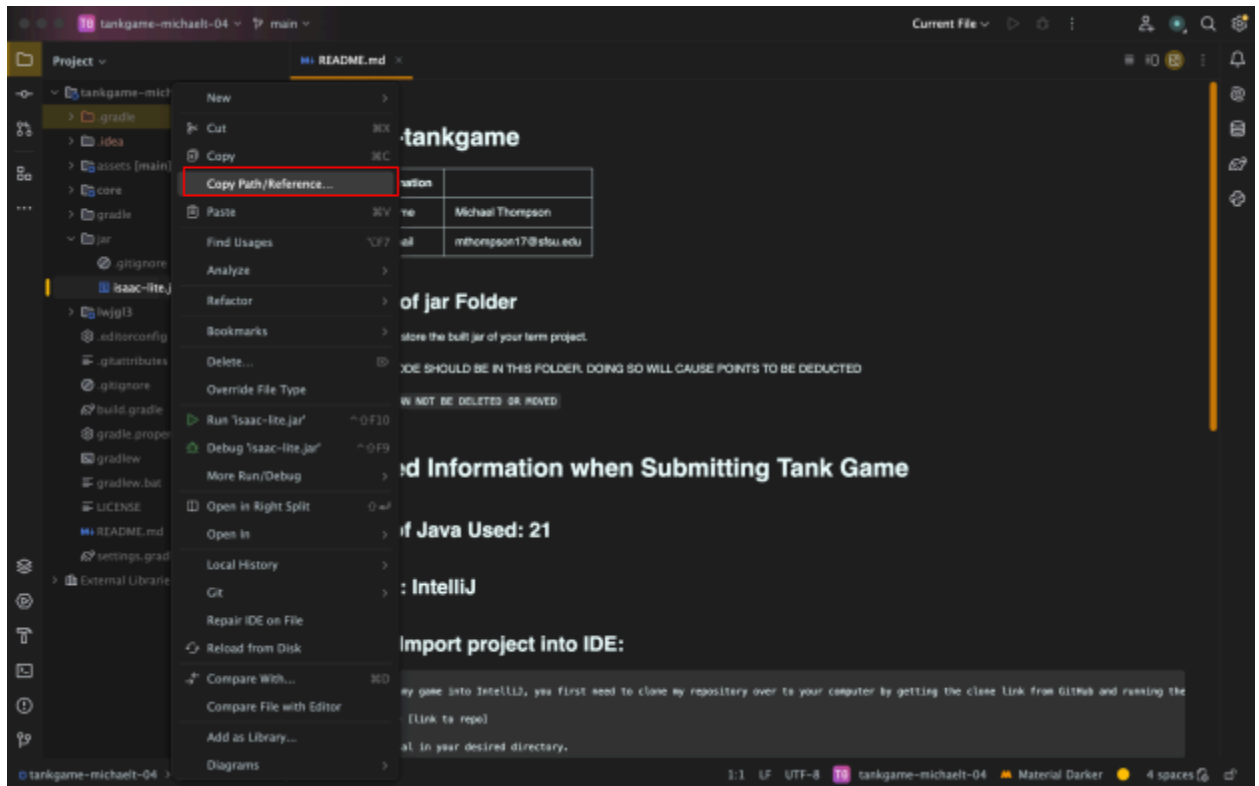


### 3.3 How to Run the JAR

- There are a couple ways to run the JAR. You can either do it through the command line using the command

```
java -jar [path to the jar]
```

You can find the path to the jar either through IntelliJ or through your file browser.



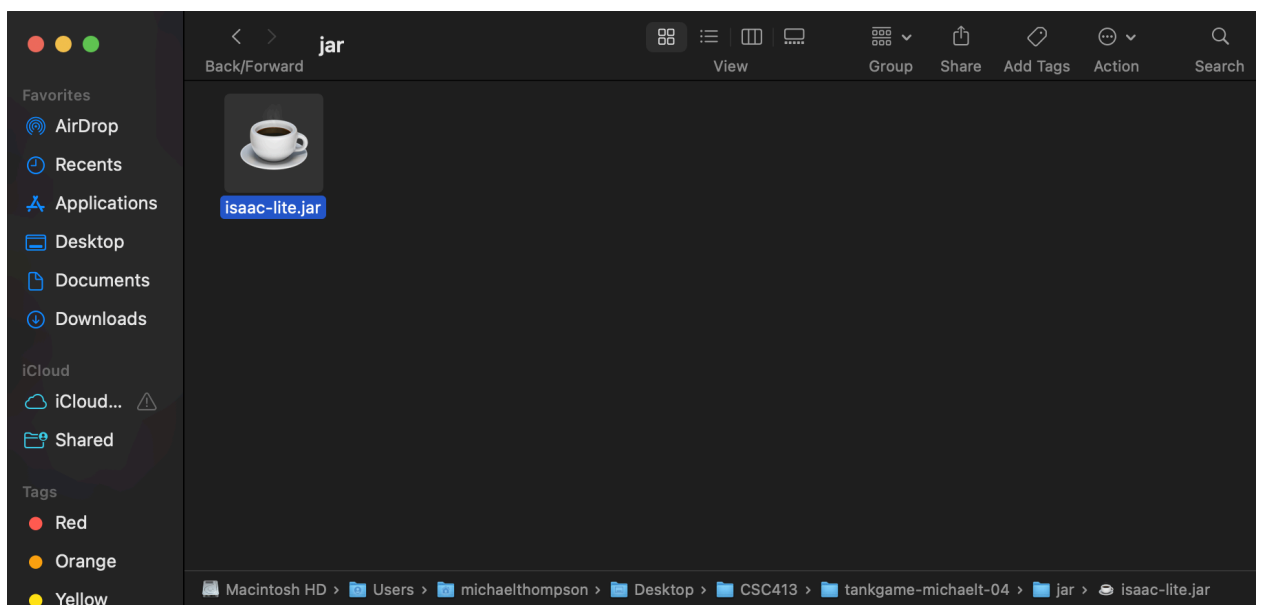
If you copy the absolute path, you can run the jar from any directory in the command line.

```
michaelthompson — -zsh — 80x24

Last login: Sat Dec 14 12:05:05 on ttys000
michaelthompson@MacBook-Air ~ % java -jar /Users/michaelthompson/Desktop/CSC413/
tankgame-michaelt-04/jar/isaac-lite.jar
```

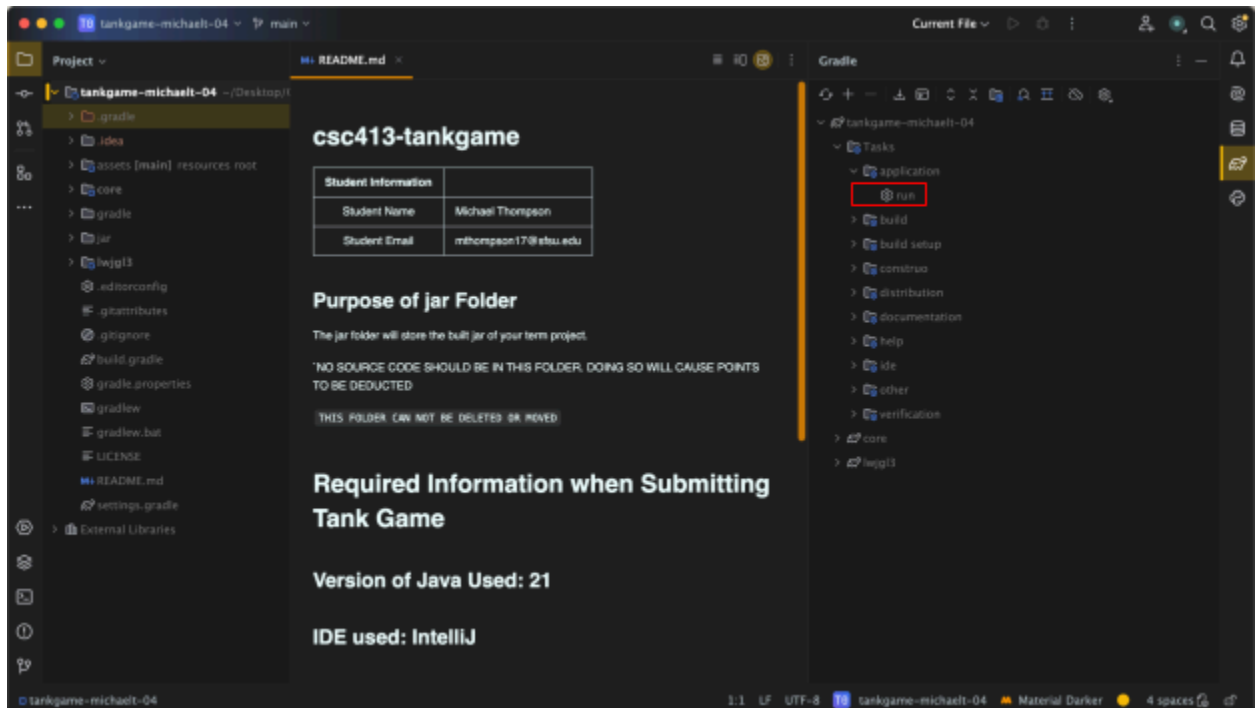
This will run the JAR file, which will start the game.

Or you can simply find the JAR file in your file browser, and double click on it to execute it.



### 3.4 How to run my game in IntelliJ

- If you want to run my game in IntelliJ without using the JAR, I listed the instructions above when talking about importing/building it in your own IntelliJ. But once you have the project imported and built correctly into IntelliJ, in the Gradle tasks you can find the “run” task in the “application” folder.



## 4 About My Game

### 4.1 Controls and Rules

- The rules and controls for my game are relatively simple.
  - Controls:
    - Movement:
      - To move around use the W A S D keys, you can move in any direction and there is no limit on your movement.
    - Shooting:
      - To shoot, use the arrow keys UP DOWN LEFT RIGHT. Your shooting is locked to the cardinal directions, you cannot shoot diagonally. This is how the original game functions.
    - Placing Bombs:

- If you have bombs, you can place them using the SPACE bar. Once you place a bomb, it will begin to flash before exploding and dealing damage to anything in its radius.
- Pause Game:
  - The player can always pause the game at any point by pressing ESCAPE. This will pause the entire game, from this screen the player can either resume, exit the game, or restart.
- Move in Menu:
  - When in any menu, the player can move the selector using the arrow keys.
- Select in Menu:
  - When in any menu, the player can select using the Enter key.
- Rules:
  - Objective:
    - This game functions like a normal roguelike. The objective is to explore the generated dungeon, and beat the boss which lies in the boss room somewhere in the dungeon. In order to get to the boss, you must go through multiple rooms of enemies and clearing the rooms as you go.
  - Health:
    - The player spawns with a default health of three hearts (6 total health points). If you get hit more times than you have health, you will die and have to restart.
      - There are a few different things that could deal damage to the player.
        - Enemies: Upon any contact with enemies the player will take damage. Some enemies fire projectiles which will deal damage on impact with the player.
        - Bombs: The player can place bombs to either break rocks or deal damage. However, this damage also applies to the player if they are in the blast radius.
        - Boss: The boss functions very similarly to the normal enemies. If the boss hits you, you will take one whole heart of damage (2 health points).
  - Combat:
    - The combat is room-based, this means that when you enter a room you are not able to leave until you have cleared all the enemies in that room. While the enemies are alive, the doors will be locked and you will not be able to go through them.
    - The player can deal damage by shooting Isaac's tears. These will deal damage to the enemies on impact.
  - Power-ups:

- There are two types of power-ups in my game. Similarly to the original game, there are pills and items.
  - Pills: Pills have a 50% chance of spawning after clearing a room of enemies. In the original game, pills can either raise or lower your stats. For my version, I made it so that the pills always give you beneficial stat changes because the game is not as long as the original. There are three different pills: health up, speed up, and tears up. Health up increases the players health, speed up increases the players movement speed, and tears up increases the players shot speed. These can compound on each other, so if you collect multiple pills in a run you can increase your stats dramatically.
  - Items: Items spawn in item rooms (highlighted with a golden door in game), and not only change something about your character but also change the appearance of the character. Some items can have affects on your stats, but others change something else. In my game I have three different items: MEAT!, Pyro, and The Inner Eye. Each of these have very different effects on the player.
    - MEAT!: This item doubles the players damage and gives more health. This is the appearance it gives Isaac:



- Pyro: This item does not effect the players stats, it gives the player 99 bombs. This is the appearance it gives Isaac:



- The Inner Eye: This item does not effect the players stats, it gives the player a triple shot. With this item, the player will shoot three tears at once. This is the appearance it gives Isaac:



- Death:
  - When the player dies, they can restart the game all over again from the beginning. This will generate a new dungeon for the player to control and reset the player back to default stats. There are no checkpoints in this game, no matter where the player is, if the player dies they must restart from the beginning.
- Restarting:
  - At any point in the game, the player is able to go to the pause screen and restart the game. This will function the same as when the player dies and restarts, a new map will be generated and the player will be reset to default stats.

## 4.2 Assumptions Made When Designing and Implementing

- I assume that the person who wants to play my game will be able to open the JAR file. This means that they have a compatible version of Java on their computer.
- I assumed that the person playing my game has played a roguelike or The Binding of Isaac previously. I did not include any instructions in my actual game, it would be a good idea in the future to add a help menu which contains the objective along with how to play the game.
- I assumed that the person who wants to play my game will have a computer than can run it. My game is not demanding in any way, but if the computer is bad enough it might lag.

## 4.3 Class Diagram and Description

- UML Class Diagram:



## Core:

- GameScreen: The main gameplay screen, updates and renders game elements. Handles different transitions for game states and manages some resources.
- GameManager: This is the main controller for game logic and mechanics. Ensures the game runs correctly and smoothly, and handles transitions between rooms.
- RoomManager: This is manager for the rooms. Creates the rooms based on the map handed to it by the MapGenerator. Ensures all rooms are connected, and that the rooms are created.
- HudManager: Manages all of the hud elements for the player. This includes the hearts, and the bomb count.

#### Environment:

- MapGenerator: Responsible for procedurally generating a grid-based map. This is essential for making a game like The Binding of Isaac, it allows for replayability and some sense of randomness. In order to implement this class, I found a good write up on how the original map generation worked for The Binding of Isaac and tried my best to implement it into my version of the game.
- Room: Creates a Room instance, the room is decided by the CSV file, it creates all of the solid tiles in the room, and handles all of the monsters/rocks/walls in the room. Manages the layout, interactions, and entities.
- Door: Doors connect the rooms to each other. This allows the player to walk between rooms and explore the maps.

#### Blocks:

- Rock: Handles all of the rock specific things, such as being breakable and having its own sprite.
- Wall: Handles all of the wall specific things, this functions the same as the outside wall of the room, but has its own sprite to know the player they cannot break it.

#### Characters:

- Character: This is a base class for all of the characters. Most of the characters share properties, this class contains all of those properties and classes. Used by Player, Monster, and Boss.
- Player: This class handles all of the player specific things. Such as the player specific sprites, animations, movements, etc.
- Monster: This is a base class for the monsters. This has some of the monster specific attributes and methods.

#### Monsters:

- Clotty: This is a specific type of monster. The Clotty class has all of the Clotty specific behavior, animations, etc. Clotty will always move towards the player, until it gets within a certain distance then it will attack. For its attack, it shoots four projectiles out (each in a cardinal direction).

- Horf: This is a specific type of monster. The Horf class has all of the Horf specific behavior, animations, etc. Similarly to Clotty, the Horf will always move towards the player, dealing damage by touching the player.

#### Boss:

- Boss: This is a base class for all Bosses. In my game I only have one boss, but if I wanted to add more I can use this class to build the other bosses off.
- Chub: This is a type of boss. Chub is the only boss I have in my game. This class has Chub specific behavior. Chub will randomly move around the room in cardinal directions only, then whenever the player aligns with Chub in a cardinal direction, it will charge toward the player. This is Chub's only way of attacking, it deals damage if it touches the player.

#### Utils:

- CollisionDetector: This class manages all of the collision between all of the entities in the game. It is able to handle character and character, character and tile, projectile and character, and projectile and tile collision. This is a key aspect of the game and very important to how the game functions.
- InputHandler: This class handles most of the inputs for the game, including character movement keys, shoot keys, and placing bomb keys. It passes back the values when a key is pressed so that other classes can handle them.

#### Items:

- Powerup: This is a base class for all powerups (pills and items). It is meant to be extended from by pill and item.
- Item: Items are powerups that can change stats and apply an appearance change to the player. Instead of making separate classes for each item, I just passed in which stat it will modify and this distinguishes which item it will be. The item is generated in an Item Room, so when an instance of the Room class is marked as an item room, an item will be generated.
- Pill: Pills are powerups that only change the stats of the player. Similarly to the item class, there are not multiple classes for the different pills, it is determined by the stat it will change. Pills can be generated after a room is cleared of enemies, it is generated by the Room class.
- Bomb: This class is responsible for all of the bomb behavior and interactions. Bombs are held by the player and placed by the player.

#### Projectiles:

- Projectile: This class is used for creating all of the projectiles in the game. This means the player's tears and the monster's projectiles. Including its size and how it acts.

#### Screens:

- StartScreen: Responsible for the start screen, manages the drawing of it as well as how the user is able to select the choices on the start screen.
- PauseScreen: Responsible for the pause screen, manages the drawing of it as well as how the user is able to select the choices on the pause screen. Allowing the user to restart, resume, or exit the game.
- EndScreen: Responsible for the end screen (screen that shows after killing boss), manages the drawing of the screen as well as how the user is able to select the choices on the end screen. Allowing the user to select either restart or exit.
- DeathScreen: Responsible for the death screen (screen that shows when the player dies), manages the drawing of the screen as well as how the user is able to select the choices on the death screen. Allowing the user to either restart or exit.

## 5 Self-reflection on the Development Process

- This project was a lot of fun to work on, and took an immense amount of time and problem solving to do. I was really excited to work on this, because in the past I really enjoyed playing The Binding of Isaac and always thought that it was a cool game. When choosing it as my version of the Tank Game, I didn't think it would be too bad because it was a relatively simple 2D dungeon crawler, however this couldnt be further from the truth. It ended up taking me a ton of time to make and a lot of research on things I haven't done before. There were many times where I was doubting my ability to be able to finish it in time, but in the end I was able to. It really helped me appreciate how difficult game development is and how time consuming doing things that you'd think are relatively easy to do can be. By the time the deadline arrived, I was overall fairly happy with the end product, but there are a couple bugs that I know are still in the game. All of the remaining issues are related to the map generation and connecting of the rooms which took a lot of the development time to figure out. Sometimes you get a map that either doesnt have an item room or boss room, or it does have one but it isn't accessible. This is a very frustrating bug, but I prioritized making sure everything else was working as I wanted it to so that I could get a mostly working game turned in on time. This bug is part of the reason why I allow the user to restart the game at any point, so that if you get a bugged map, you can just regenerate a new one and play again.

Overall I am very happy with this project and enjoyed it a lot. I plan to continue building on my game and hopefully adding more things from the original game, and polishing what I currently have.

## 6 Project Conclusion

- My game fulfills all of the requirements and does a fairly good job at emulating the original game that I had hoped to recreate. There are no major bugs which cause the game to crash, it runs smoothly, and the player gets the feeling of playing the original game, which was one of the most important things to me. This was a great experience, I enjoyed this project a lot and learned so much from it.