CSC 413 Project Documentation

Summer 2022

Timmy Tram

921102465

CSC413.01

https://github.com/csc413-SFSU-Souza/csc413-tankgame-TimmyTram

Table of Contents

1 Introduction

1.1 Project Overview

The goal of this project was to create a game where players get to play as tanks and shoot each other until a player wins the game. To achieve this, we must use various object-oriented programming concepts that we learned over the course of this summer semester to create a clean and maintainable codebase open to extension for future features if we so wanted.

1.2 Introduction of the Tank game

The tank game is a game where 2 players play as tanks and shoot each other until a player wins. The game will contain a small arena with interactable walls that may or may not break depending on the rock type. The game also contains several power ups which have passive buffs for the player.

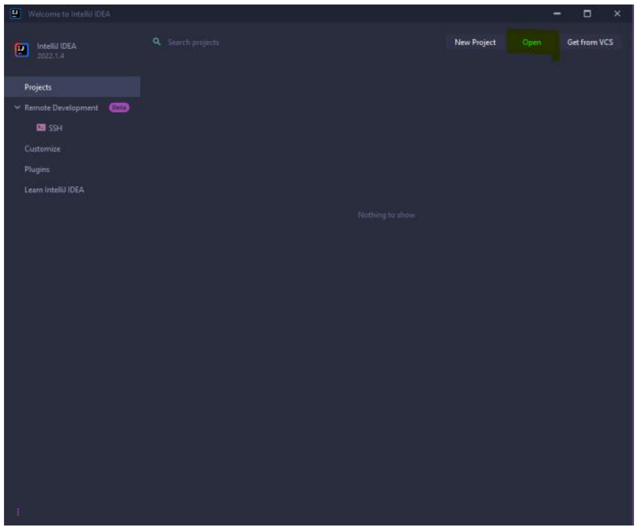
2 Development Environment

a. Version of Java used: openjdk-16 or Java Version 16.0.2
b. IDE used: IntelliJ IDEA Ultimate Edition 2022.1.2

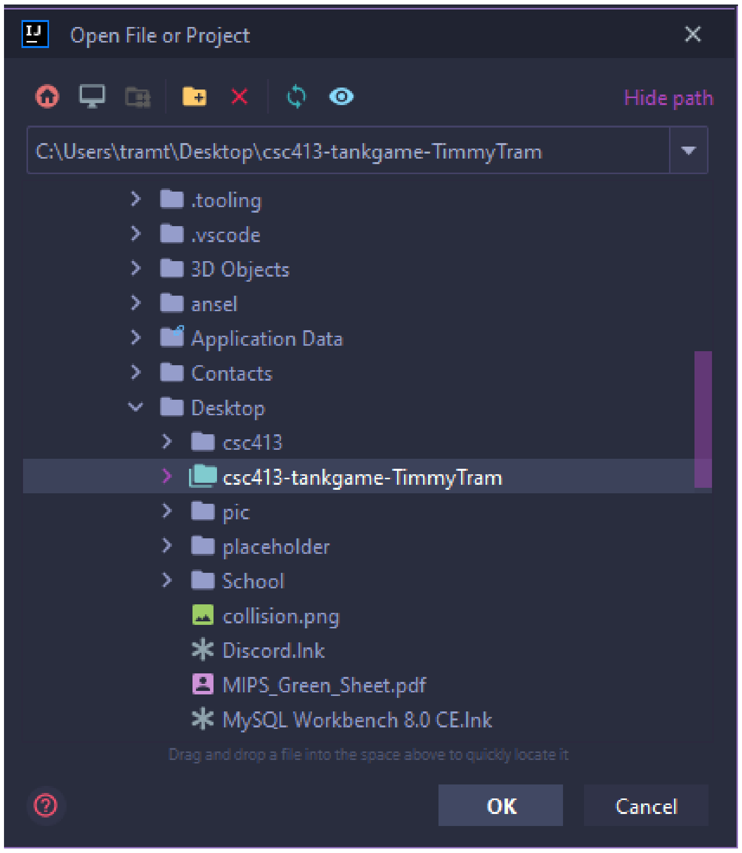
3 How to Build/Import your Project

3.1 Importing the Project

1. git clone https://github.com/csc413-SFSU-Souza/csc413-tankgame-TimmyTram.git
2. In IntelliJ IDEA select OPEN

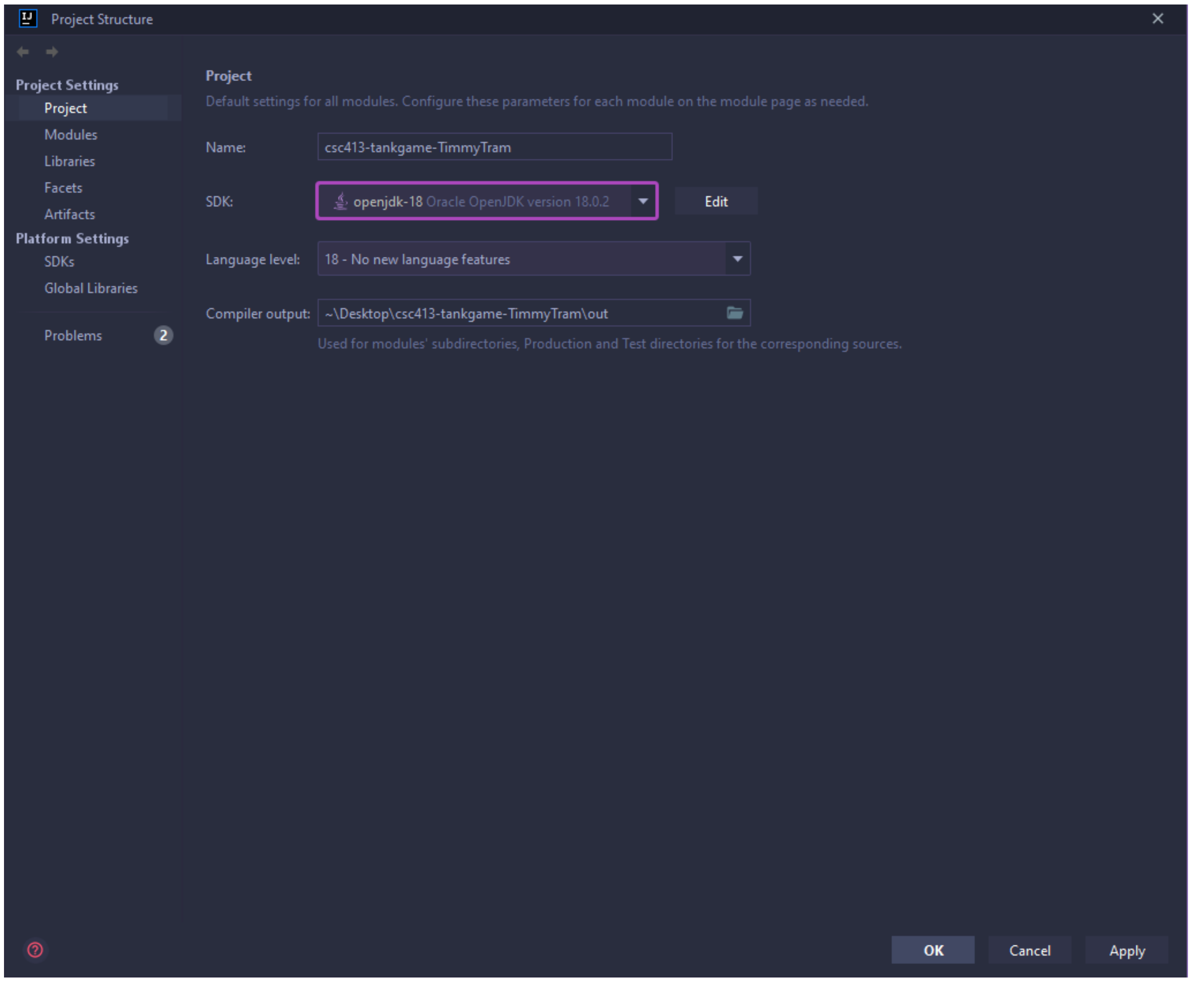


3. Select csc413-tankgame-TimmyTram



3.2 Configuring the Project

1. Open the Project Structure under the File Button (CTRL+ALT+SHIFT+S).
2. For SDK select any SDK that is at least Java 16 or higher and same for Language Level then hit APPLY and OK.


Project Structure

Project Settings
Project
Modules
Libraries
Facets
Artifacts
Platform Settings
SDKs
Global Libraries

Problems 2

Project

Default settings for all modules. Configure these parameters for each module on the module page as needed.

Name: csc413-tankgame-TimmyTram

SDK: openjdk-18 Oracle OpenJDK version 18.0.2

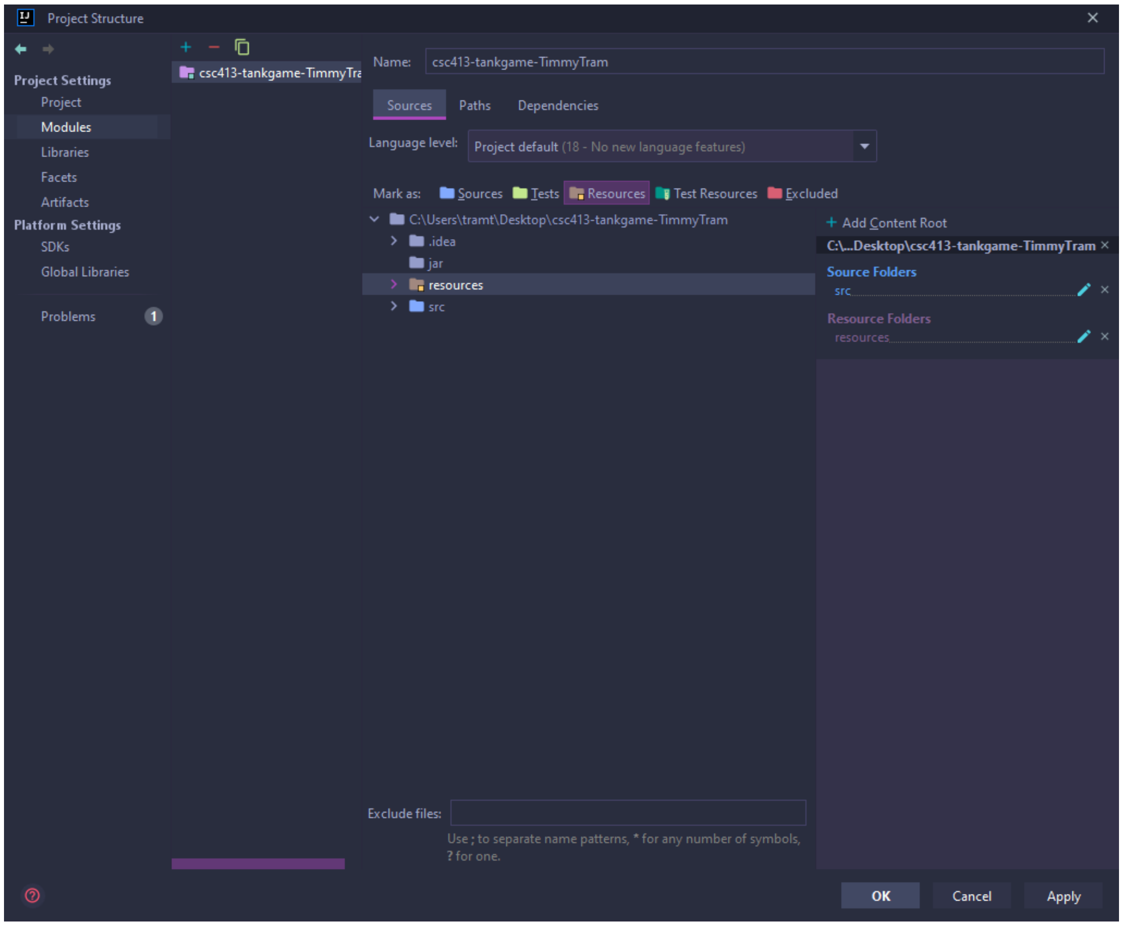
Language level: 18 - No new language features

Compiler output: ~\Desktop\csc413-tankgame-TimmyTram\out

Used for modules' subdirectories, Production and Test directories for the corresponding sources.

3. In Project Structure and under Project Settings click modules.

4. Make sure to mark the src folder as a source folder and the resource folder as Resources and then hit APPLY and OK.


Project Structure

Name: csc413-tankgame-TimmyTram

Language level: Project default (18 - No new language features)

Mark as: Sources Tests Resources Test Resources Excluded

C:\Users\tramt\Desktop\csc413-tankgame-TimmyTram
- .idea
- jar
- resources
- src

C:\...Desktop\csc413-tankgame-TimmyTram

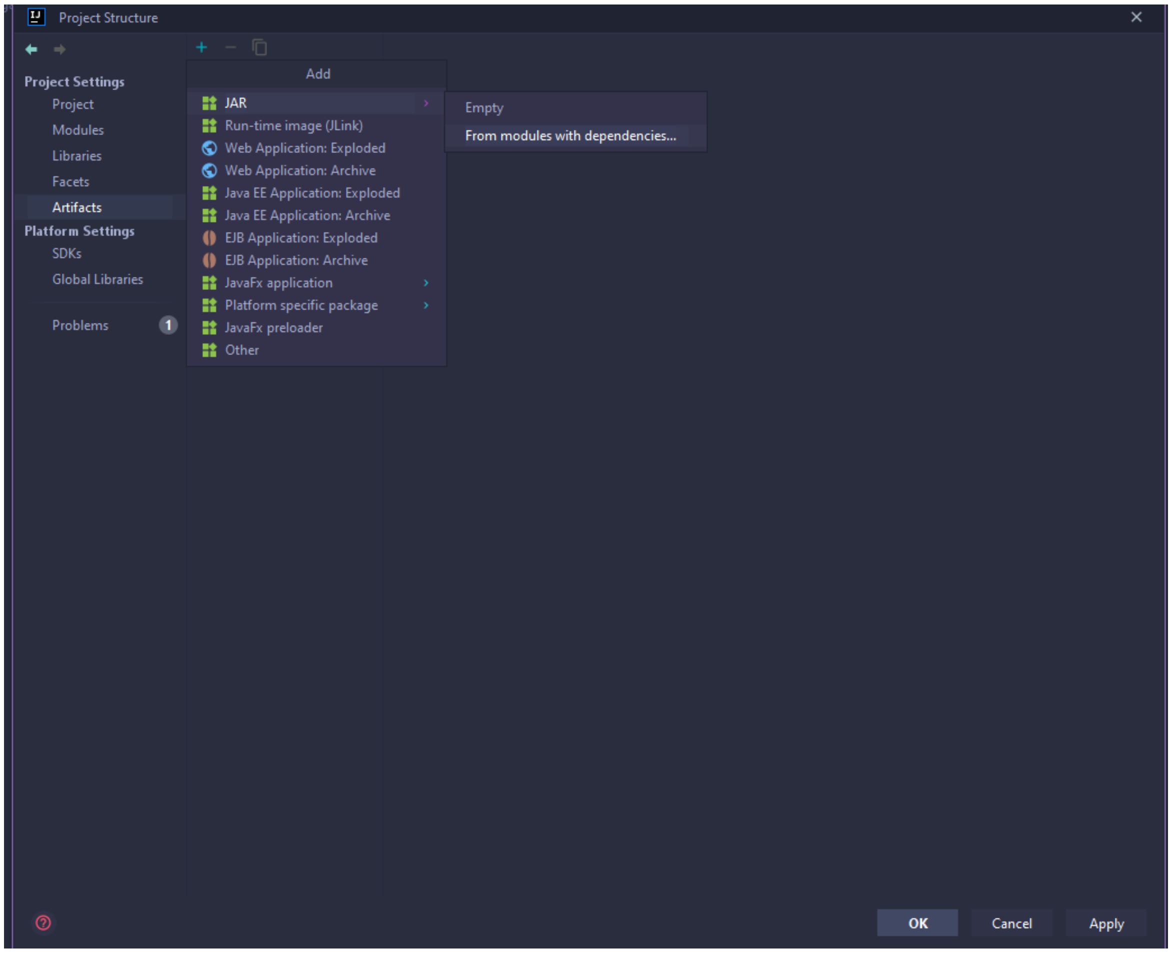
Source Folders
src

Resource Folders
resources

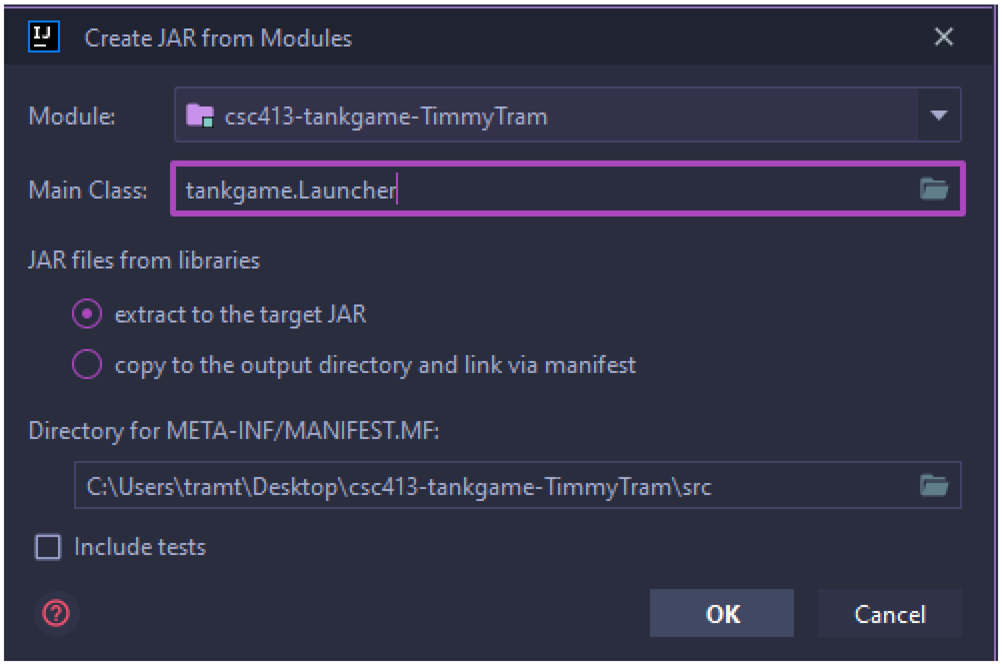
Exclude files:

Use ; to separate name patterns, * for any number of symbols, ? for one.

5. Then under Project Setting select Artifacts and press the + button to create a JAR from modules with dependencies.



6. For Main Class select Launcher.java or type tankgame.Launcher and then hit OK for everything


Create JAR from Modules

Module: csc413-tankgame-TimmyTram

Main Class: tankgame.Launcher

JAR files from libraries

[✓ extract to the target JAR]

copy to the output directory and link via manifest

Directory for META-INF/MANIFEST.MF:

C:\Users\tramt\Desktop\csc413-tankgame-TimmyTram\src

Include tests

3.3 Building the Project

1. Under Build button on the top just hit Build the out folder
2. Then if you want to build the JAR press build → Build Artifact → Build

4 How to Run your Project

1. Double click the JAR file to run the game
2. Open a console located in the same directory as the jar and type → java -jar csc413-tankgame-TimmyTram.jar
3. Run via the IDE by pressing the play button in IntelliJ or right clicking on Launcher.java and selecting RUN

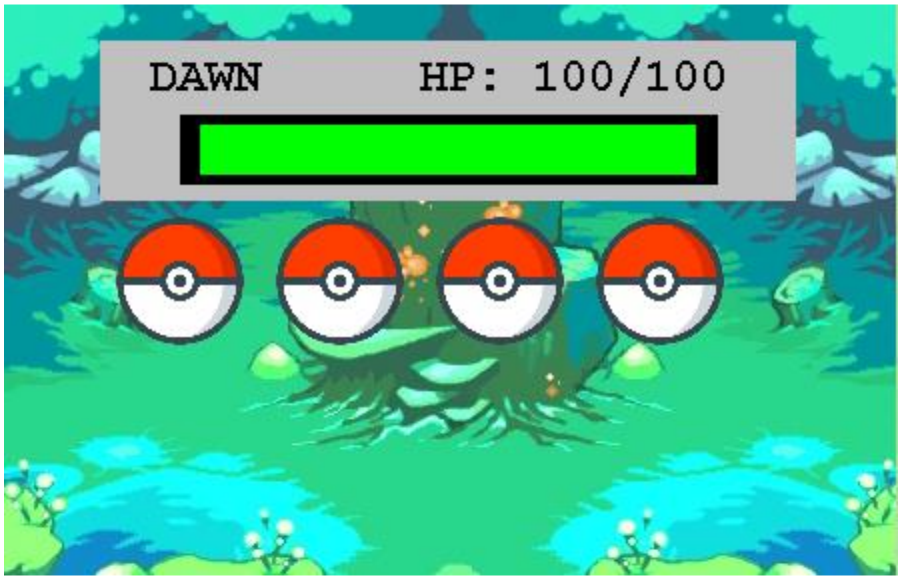
5 Rules and Controls

5.1 Rules

Each player has 100 health points and has 4 extra lives meaning each player has a total of 5 lives.

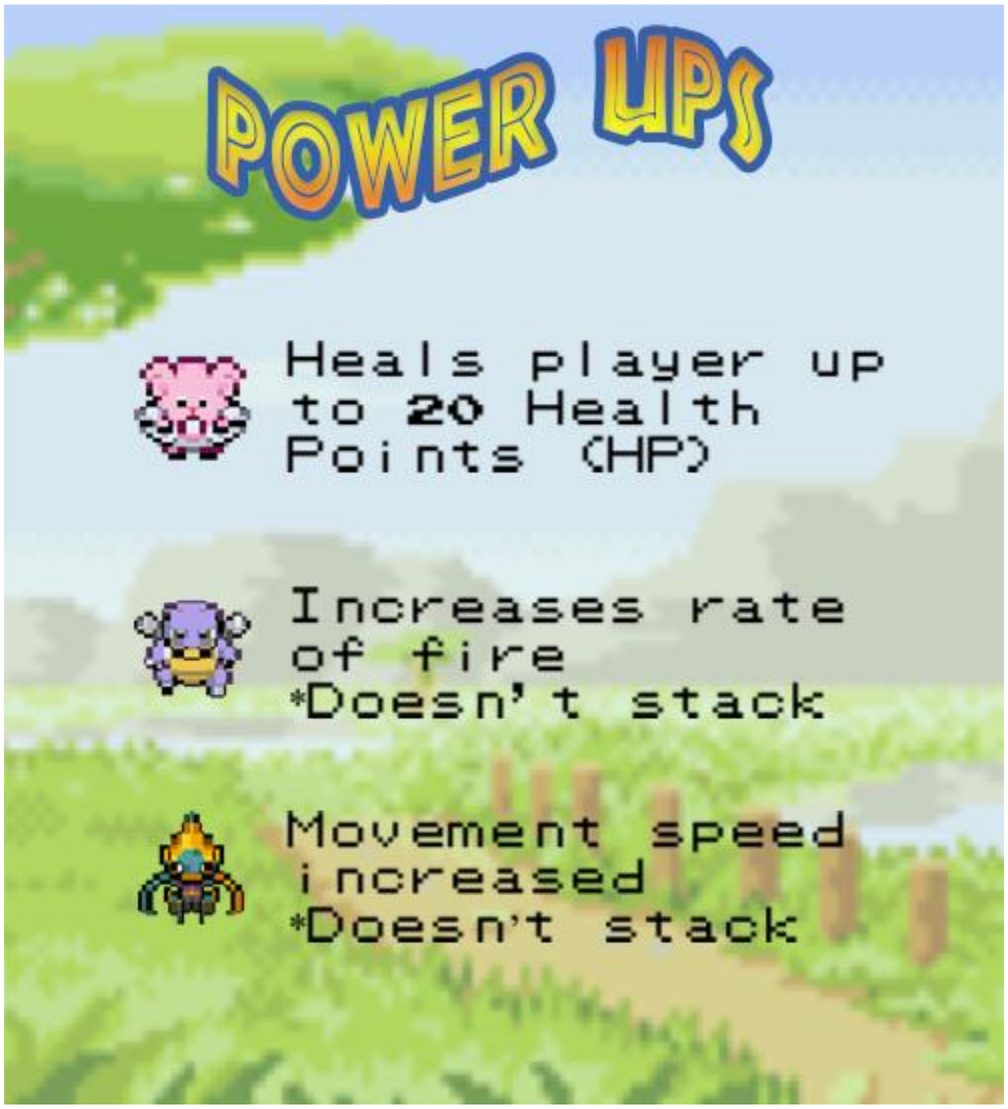
First player to lose all their lives loses the game.

Instead of tanks, one player plays as a Pokémon trainer and the other player plays as the actual Pokémon. The player image has no affect on gameplay at all.



The power ups in the game are as follows:

- HEAL → Restores up to 20 health points for the tank that picks up the power up.
- BARRAGE → Increases the rate of fire of the tank that picks up the power up. (DOES NOT STACK)
- SPEED BOOST → Increases the movement speed of the tank that picks up the power up. (DOES NOT STACK)
- These options may be reviewed in-game via the help menu if needed.



POWER UPS

Heals player up to 20 Health Points (HP)

Increases rate of fire
*Doesn't stack

Movement speed increased
*Doesn't stack

5.2 Controls

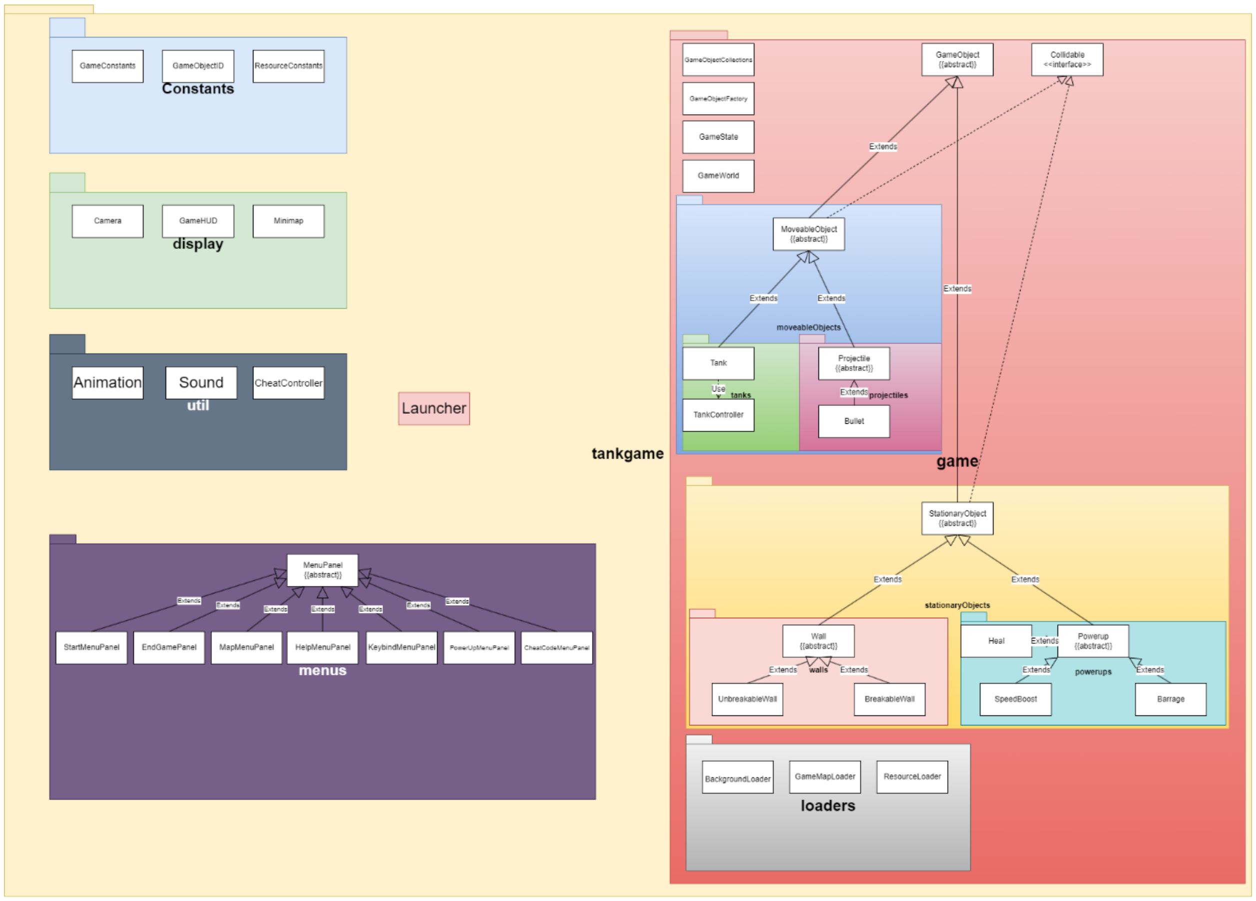
TANK CONTROLS	PLAYER 1	PLAYER 2
FORWARD	W	UP ARROW
BACKWARD	S	DOWN ARROW
ROTATE LEFT	A	LEFT ARROW
ROTATE RIGHT	D	RIGHT ARROW
SHOOT	SPACEBAR	ENTER

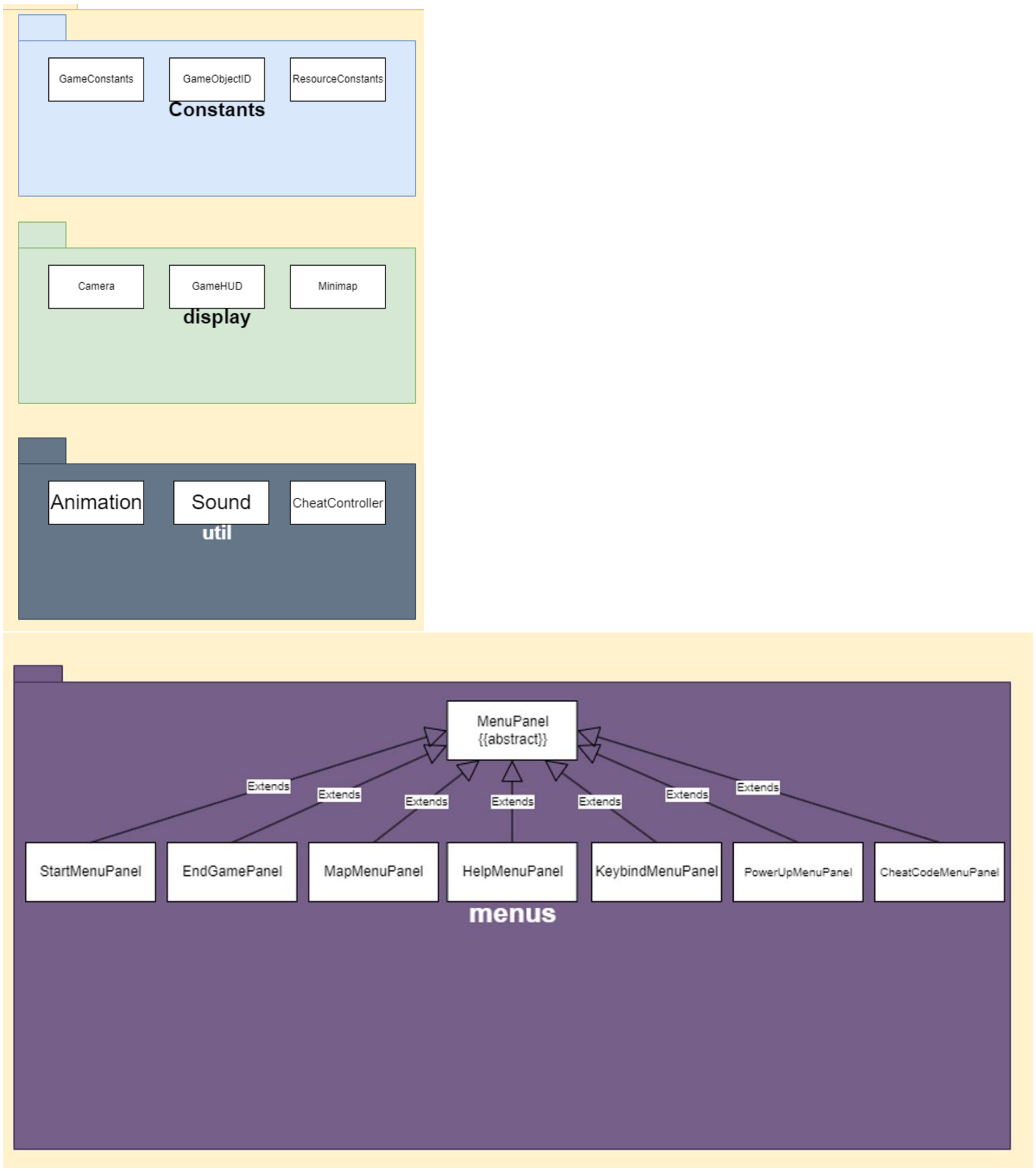
CHEAT CONTROLS	KEYS
TOGGLE HITBOXES	F1
TOGGLE ONE SHOT ONE KILL MODE	F2

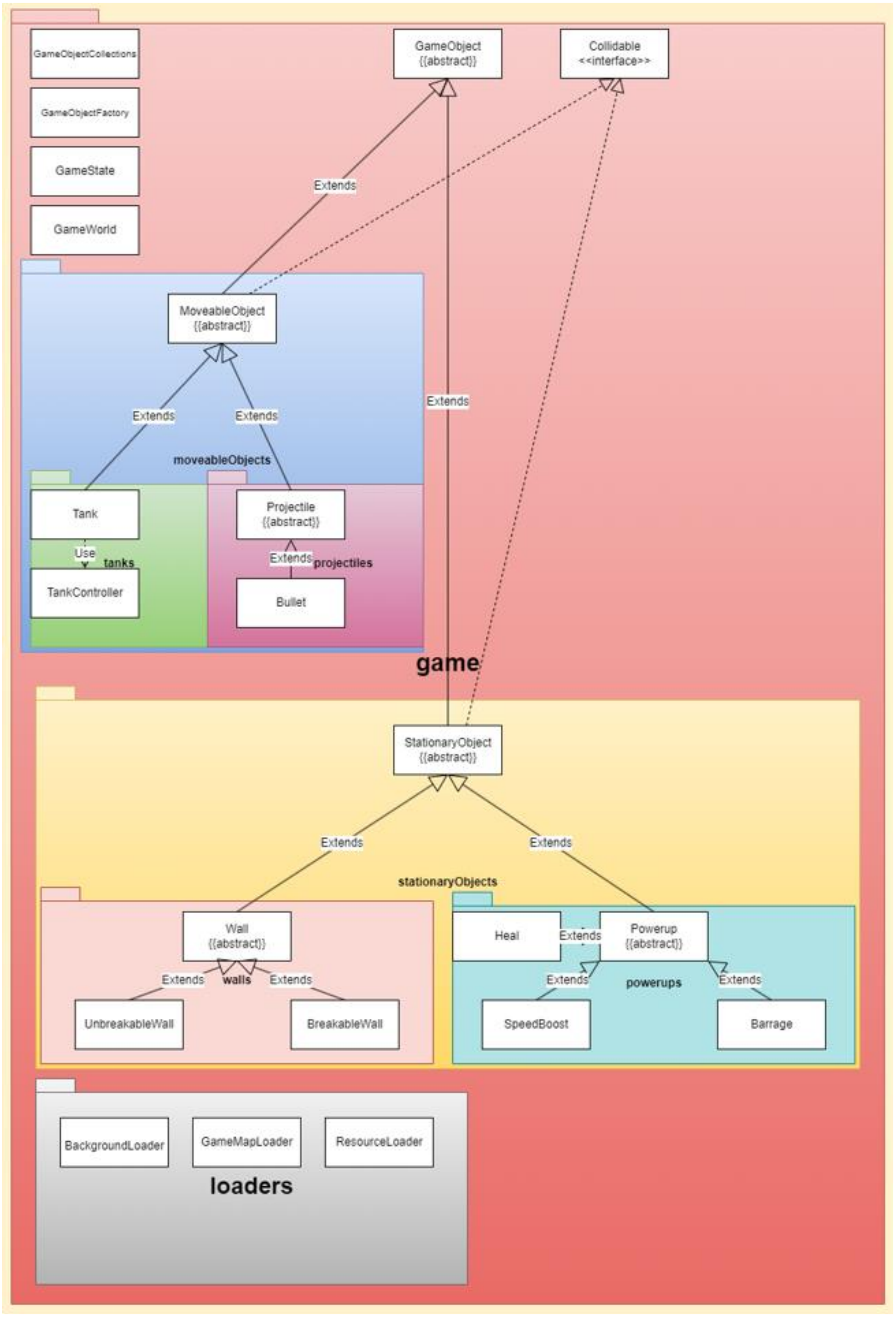
6 Assumptions Made

I assumed the player of this game would have at least a stable computer with a resolution greater than or equal to 1280 by 960.

7 Tank Game Class Diagram




GameConstants
GameObjectID
ResourceConstants
Constants
Camera
GameHUD
Minimap
display
Animation
Sound
CheatController
util
MenuPanel
{{abstract}}
Extends
StartMenuPanel
EndGamePanel
MapMenuPanel
HelpMenuPanel
KeybindMenuPanel
PowerUpMenuPanel
CheatCodeMenuPanel
menus


GameObjectCollections
GameObjectFactory
GameState
GameWorld
GameObject {{abstract}}
Collidable <<interface>>
Extends
MoveableObject {{abstract}}
Extends
Extends
Extends
moveableObjects
Tank
Use
tanks
TankController
Projectile {{abstract}}
Extends
projectiles
Bullet
game
StationaryObject {{abstract}}
Extends
Extends
stationaryObjects
Wall {{abstract}}
Extends
walls
Extends
UnbreakableWall
BreakableWall
Heal
Extends
Powerup {{abstract}}
Extends
powerups
Extends
SpeedBoost
Barrage
BackgroundLoader
GameMapLoader
ResourceLoader
loaders

8 Class Descriptions

8.1 Tankgame Package

The tankgame package is the main package that contains all other sub-packages. It contains a single class, the Launcher which is the main entry point of this program.

- **Launcher:**
  - Initializes all menuPanels and deals with changing the current panel to different panels.

8.2 Constants Package

The constants package contains 3 classes each with containing constants related to their class name.

- **GameConstants:**
  - This class contains constants related to what the size each menu should be as well as containing hex codes for the colors of the buttons.
- **GameObjectID:**
  - This class contains constants which we use to identify what kind of game object we should create when parsing the csv file for maps.
- **ResourceConstants:**
  - This class contains constants which we use to hold the filenames of assets for the game. It is also used as the keys for getting resources from the ResourceLoader.

8.3 Util Package

The utility package contains miscellaneous classes that could not be sorted into smaller sub packages.

- **Animation:**
  - This class handles the animations given a list of bufferedImages.
- **CheatController:**
  - This class implements a KeyListener and is only initialized when the game starts.
  - If F1 or F2 is pressed it requests the GameState object to move onto its next state
  - Prints current state of cheats to the console.
- **Sound:**
  - This class deals with playing sounds for the game as well as music.
  - Can loop music continuously and stop when the game is in a stopped game state.

8.4 Menus Package

The menus package contains multiple panels that represent the different windows.

- **MenuPanel:**
  - An abstract class that encapsulates shared code between all menu panels such as loading the correct background.
  - createLabel is a function that creates a reusable label component to avoid repeating unnecessary code that all sub-classes of MenuPanel can use.
  - createButton is a function that creates a reusable label component to avoid repeating unnecessary code that all sub-classes of MenuPanel can use.
- **StartMenuPanel:**
  - Extends MenuPanel and creates 4 buttons.

        - START → Selects a random map and starts the game.
        - MAPS → Sets the current panel to MapMenuPanel.
        - HELP → Sets the current panel to HelpMenuPanel.
        - EXIT → Closes the game.
- **MapMenuPanel:**
    - Extends MenuPanel and creates 4 buttons.
        - BACK → Returns to the StartMenuPanel.
        - 2fort → Selects 2fort.csv as the current map.
        - Pillar → Selects pillars.csv as the current map.
        - Tunnels → Selects tunnels.csv as the current map.
- **HelpMenuPanel:**
    - Extends MenuPanel and creates 4 buttons.
        - KEYBIND → Sets the current panel to KeyBindMenuPanel.
        - POWER UPS → Sets the current panel to PowerUpMenuPanel.
        - CHEAT CODES → Sets the current panel to CheatCodePanel.
- **KeyBindMenuPanel:**
    - Extends MenuPanel and creates 18 labels that represent a table of what the key binds are for each player to control their tank. Also creates a single button to return to the HelpMenuPanel.
- **PowerUpMenuPanel:**
    - Extends MenuPanel and creates a single button to return to the HelpMenuPanel.
    - The background is set to a predetermined asset that lists what power up looks like and what it does exactly.
- **CheatCodeMenuPanel:**
    - Extends MenuPanel and creates 9 labels that represents a table of what the key binds are to cheat / debug. Also creates a single button to return to the HelpMenuPanel.
- **EndGamePanel**:
    - Extends MenuPanel.
    - Creates a label that shows which player won.
        - Uses updateWinnerStatus() to help determine which player won.
        - Uses updateWinPotrait() to get the asset of the player character.

8.5 Display Package

The display package contains code and classes related to how the game is rendered or what needs to be rendered to the screen.

- **Camera:**
    - Constructor takes a tank which it will track, and another parameter called hud limit which is uses to tell how far it should render / draw the screen to avoid drawing over the GameHUD.
    - drawSplitScreen takes in the world as a parameter and uses it to calculate how far to draw the split screen.
    - getSplitScreen returns the split screen buffered image for use.
    - checkBorderX does various calculations to avoid going outside the raster on the x-axis.

    - checkBorderY does various calculation to avoid going outside the raster on the y-axis as well as avoid drawing over the gameHUD.
- **GameHUD:**
    - Constructor takes in a tank to track its health points and lives counter as well as positional arguments to help determine where to draw the HUD and a background image to spice up the HUD.
    - drawHUD takes in a Graphics2D object to draw the health bar and lives counter.
    - drawLives takes in a Graphics2D object and gets the lives of the tank it is tracking and calculates where to place the lives counter in a scalable manner by using a for loop.
    - drawHealthBar takes in a Graphics2D object and gets the health points of the tank it is tracking as well as the name of tank. It then draws a health bar to the screen using rectangles with the width of the rectangle being equivalent to the amount of health points the tank has relative to its tank's maximum health points possible. Additionally, the color of the health bar changes every 33% of health lost going from green to yellow to red.
- **Minimap:**
    - Constructor calculates the width and height of the mini map.
    - drawMinimap takes in a BufferedImage of the world and a Graphics2D object as parameters. Calculates the size of the minimap using the BufferedImage of the world and scaling it down to be 20% of the original world size.
    - getScaledWidth allows user to get the true width of the mini map.
    - getScaledHeight allows user to get the true height of the mini map.

8.6 Game Package

The game package contains code and classes related to all game objects or entities as well as the creation of game objects.

- **GameObject:**
    - An abstract class that provides the ideas of what a GameObject should be.
    - Constructor takes in positional parameters x and y as well as a BufferedImage to render the tank.
    - Provides an abstract void drawImage method to allow each sub-class of GameObject to provide their own implementation of how they should render themselves.
    - Provides an abstract void drawHitbox method to allow each sub-class of GameObject to provide their own implementation of how they should render their hitboxes.
    - getHitbox simply returns the bounds of the hitbox rectangle.
    - getX returns the x position of the GameObject.
    - getY returns the y position of the GameObject.
    - Provides an abstract void update method to allow each sub-class of GameObject to provide their own implementation of what it should do when updated.
- **Collidable:**
    - An interface that provides methods of what collidable objects should do.
    - getHitBox allows the user to define what hitbox rectangle they should return.
    - handleCollision takes in another collidable object as a parameter and should be used to help process the logic when two collidable objects interact with each other.

    - isCollidable simply returns if the collidable object is collidable or not.
- **GameObjectCollections:**
    - A data structure that is a wrapper for an ArrayList.
    - Uses Java Generics to ensure that it will only store GameObjects or sub-classes of GameObjects.
    - Created specifically to encapsulate away the update for loop and draw for loop so that the update loop in GameWorld is cleaner.
    - Cannot use for-each loops in the update and draw method or else we will get a ConcurrentModificationException due to how iterators work and adding new GameObjects to the list.
- **GameObjectFactory:**
    - Uses a creational design pattern called the factory pattern to help with the creation of GameObjects.
    - Provides a single method for usage called createGameObject that takes in an id, positional arguments and the GameWorld as parameters.
        - Uses the GameObjectID from constants package as case statements for the switch statement to allow easier renaming of switch cases from the constants package.
        - Allows the creation of:
            - Tank player 1
            - Tank player 2
            - Border Unbreakable walls (Collisionless)
            - Unbreakable walls (Collidable)
            - Breakable walls
            - Heal power ups
            - Barrage power ups
            - Speed boost power ups
        - If the id provided is null, empty meaning no characters, or is provided the id EMPTY the word, then we return null as we should create no object at a certain position.
        - If it faces an argument, it does not recognize it will throw an IllegalArgumentException
- **GameState:**
    - Contains 3 static fields to help track the state of the game.
    - Contains 3 Enum classes used as state machines to help track the state of the game.
    - These enums simply contain ON and OFF or RUNNING and STOPPED but is open to add more states such as PAUSED if we wanted to add a pause state into this game.
    - This class is mainly used by CheatController in the util package to help toggle hitboxes on and off as well as one shot one kill mode.
    - Is also used by GameWorld to help restart the game and show the end screen.
- **GameWorld:**
    - The main brains behind the game or central class.
    - Constructor calls ResourceLoader's initialization methods to simply call it once and prevent added more unneeded images into memory.

- run() method checks if the game is resettable and starts the threads for music.
- run() method updates all moveable objects as well as checks collisions and deletes GameObjects marked for deletion.
- resetGame() method resets the game and clears the GameObjectCollections lists instead of reinitializing everything.
- InitializeGame() method initializes all GameObjects, the map, the background, the huds, and controllers.
- paintComponent() method simply calls the draw methods for all relevant objects such as moveableObjects, stationaryObjects, collisionless objects, the split screens, mini map and huds.
- Provides 3 methods that allows other classes to add GameObjects to its respective GameObjectCollections lists:
    - addToMoveableGameObjectCollections allows classes such as the GameMapLoader to add MoveableObjects to the GameWorld.
    - addToStationaryGameObjectCollections allows classes such as the GameMapLoader to add StationaryObjects to the GameWorld.
    - addToCollisionlessGameObjectCollections allows classes such as the GameMapLoader to add GameObjects that do not need to be included in the checkCollision() method of GameWorld to optimize the game.
- selectMap() allows the user to select what map they want to play. Has to called resetGame() due to how we call InitializeGame() in the Launcher class. This means this method could be improved.
- checkCollision() is a private method that compares all moveableObjects against other all other moveableObjects as well as all other stationaryObjects and calls their respective handleCollsision() methods to do collision checking. Runs in O(n^2).
- deleteGarbage() is a private method that checks all moveableObjects and stationaryObjects if they are to be deleted and remove them from their respective lists.
- loadMap() is a private method that checks if a gameMap is selected and if not then it will pick a random. If provided a GameMap it will initialize that specific map.
- initTanks() is a private method that extracts the tanks from the moveableObjectGameObjectCollections to pass to other methods such as TankControllers.
- initHUD() is a private method that initializes the mini map, the cameras, and gameHUDs.
- initControllers() is a private method that creates the TankControllers for each tank player.

8.7 Loaders Package

The loaders package contains large amounts of code related to reading from the resources folder only once.

- **BackgroundLoader:**
    - Uses the singleton creational design pattern and more specifically uses lazy initialization to only create this object when it is needed.
        - There should only ever be one instance of a background which is why I chose this class to be singleton.

    - initializeBackground() is a method that is required to be used to define how much the background it should draw until it reaches the border of the world.
    - drawImage() is a method that simply draws the tiles that represents the background.
- **GameMapLoader:**
    - Uses the singleton creational design pattern and more specifically uses lazy initialization to only create this object when it is needed.
    - getEmptySpaces() is a method that returns a list of locations where there is no GameObjects.
    - initializeMap() is a method that takes in a GameWorld object and a string that represents a map.
        - Reads from a chosen csv that represents where GameObjects should be created in the GameWorld.
        - Reads the csv in an unconventional way by iterating through columns first then by row. This is to prevent the map from being rotated by 90 degrees for some reason.
        - Uses regex and forces ids in the csv to all uppercase to validate the data.
        - Passes the validated id to the GameObjectFactory as well as the row, column and GameWorld to get back a GameObject.
        - Uses if statements to add the GameObject to their respective lists.
- **ResourceLoader:**
    - This class could technically also be a singleton since we only want to instantiate this object once and call its method once.
    - Provides methods to get sounds, animations, images, and maps from the resource folders.
    - The resource loader's init methods simply gets the assets from the resources folder and stores it into memory.
    - To access certain assets from the ResourceLoader we use the ResourceConstants as keys to the ResourceLoader. This means if we wanted to change the key's name or values, we can just refactor the name of the keys in the ResourceConstants folder and help prevent us from going through every class to change the keys passed.
    - Contains private method to help read BufferedImages or AudioInputSystems in a much easier way.

8.8 MoveableObjects Package

The moveable objects package contains classes related to GameObjects that are allowed to move such as tanks and projectiles or bullets.

- **MoveableObject:**
    - An abstract class that extends the GameObject class and implements Collidable.
    - Contains 5 methods relevant to MoveableObjects.
        - setVx() simply sets the velocity in the x-direction.
        - setVy() simply sets the velocity in the y-direction.
        - getVx() returns the velocity in the x-direction.
        - getVy() returns the velocity in the y-direction.

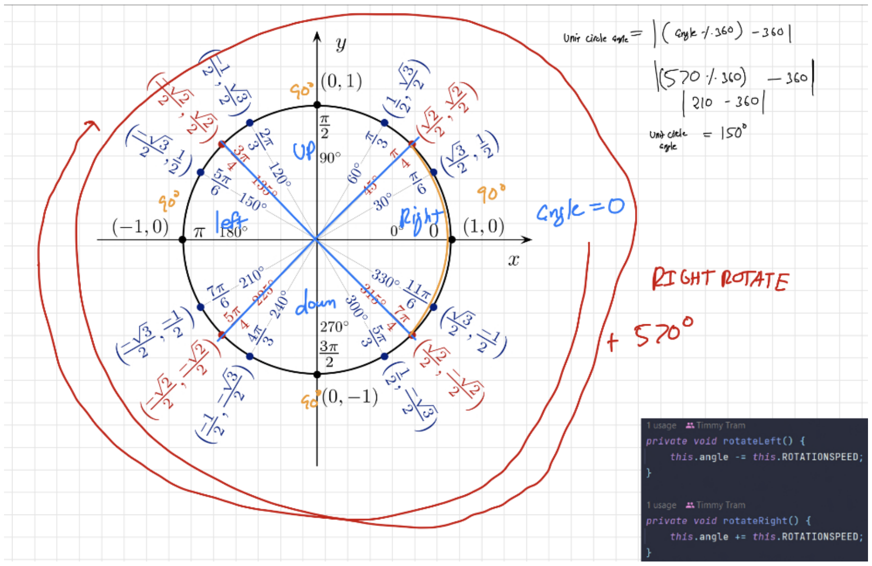
        - drawHitbox() checks if the GameState allows the GameWorld to draw the hitboxes surrounding each MoveableObject. MoveableGameObject hitboxes are represented by yellow borders.

8.8.1 Tank Package

The tanks package contains code related to tanks.

- **Tank:**
    - Constructor takes 9 parameters:
        - x, y → positional arguments that determines where to place the tank initially.
        - vx, vy → sets the initial speed of the tank.
        - angle → determines where the tank faces initially.
        - img → gives the tank an img initially.
        - playerID → gives the tank and Identification number that is used for initialization of animations.
        - name → gives the tank a name that can be used for gameHUD's healthbar.
        - gw → gives the tank a way to interact with the GameWorld
    - initAnimation() checks the playerID and gives the tank their respective animations depending on the number.
    - initBullet() checks the playerID and gives the tank their respective bullet spawning sounds, collision sounds and bullet images depending on which player they are.
    - getBulletCollideSound() returns the what the bullet sound should be when it collides.
    - setX() allows user to set where the tank's x position is. (Unused)
    - setY() allows user to set where the tank's y position is. (Unused)
    - setPosition() allows user to set the x and y position of the tank and the location of the hitbox of the tank.
    - setValidSpawnLocations() sets where a tank can respawn on the world based on the emptySpaces list from the GameMapLoader.
    - Toggle and Untoggle methods:
        - toggleUpPressed() used by tank controller to move tank upwards.
        - toggleDownPressed() used by tank controller to move tank downwards.
        - toggleRightPressed() used by tank controller to rotate the tank to the right.
        - toggleLeftPressed() used by tank controller to rotate the tank to the left.
        - toggleShootPressed() used by tank controller to create bullets.
        - unToggleUpPressed() used by tank controller to stop moving the tank upwards.
        - unToggleDownPressed() used by tank controller to stop moving tank downwards.
        - unToggleRightPressed() used by tank controller to stop rotating the tank to the right.
        - unToggleLeftPressed() used by tank controller to stop rotating the tank to the left.
        - unToggleShootPressed() used by tank controller to stop creating bullets.
    - update() checks what direction the tank should move and if the tank is allowed to shoot or if its still on cooldown. It also checks where to set the location of the animations and

what direction the animation should be faced. It also checks if cheat codes are activated or not and if the tank is alive and if it should collide with the world border.

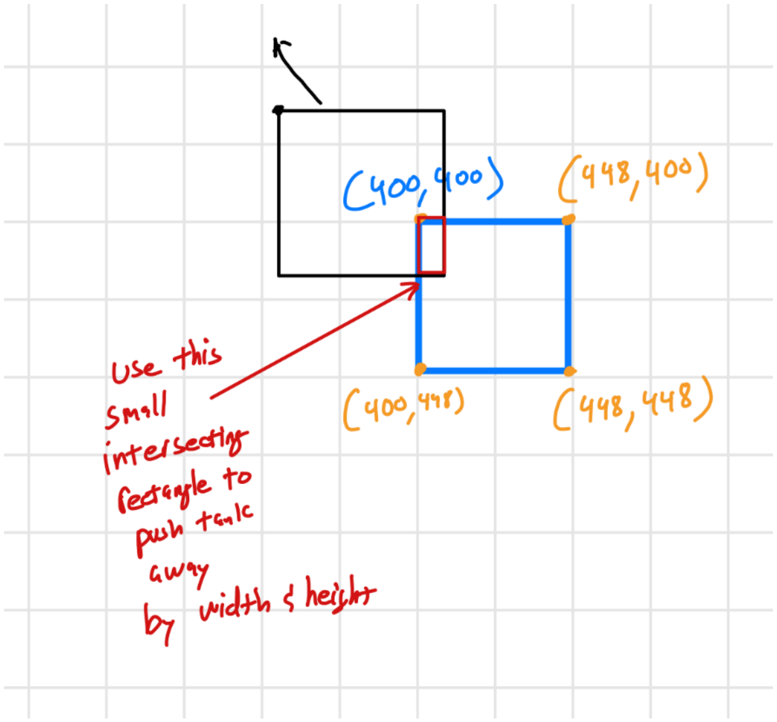
- animationHandler() is a private method that uses the angle of the tank to calculate where to the animations should face by using the unit circle as a reference.
    - Provided below is an image of how the unit circle is divided up to deal with where the animations should face.
    - Due to how rotateLeft() and rotateRight() work we need to use a little bit of math to transform the tank's angle into an angle that works with respect to this unit circle.
    - 

```
1 usage   Timmy Tram
private void rotateLeft() {
    this.angle -= this.ROTATIONSPEED;
}

1 usage   Timmy Tram
private void rotateRight() {
    this.angle += this.ROTATIONSPEED;
}
```

- rotateLeft() mutates the tank's angle by subtracting to it.
- rotateRight() mutates the tank's angle by adding to it.
- moveBackwards() moves the tank backwards while checking the border and moving the hitbox along with the tank.
- moveForwards() moves the tank forwards while checking the border and moving the hitbox along with the tank.
- shoot() creates a new bullet with a reference to the tank so the bullet knows which tank owns the bullet and prevents the newly spawned bullet from colliding with the tank it spawn on top of. Uses the GameWorld object to add the newly created bullet to the moveableGameObjectCollections.
    - This means the GameWorld is the object that owns the projectiles technically instead of having the tank own a list of bullets.
    - I chose to have the GameWorld own the projectiles because it felt it made collision checking much easier, but the tradeoff was that it made the deletion of those projectiles a bit more difficult.
- checkAlive() simply checks the health points and lives the tank has left and if both are depleted to 0 then the tank is a loser.
- randomizeSpawnLocation() uses the random class in order and the validSpawnLocations list in order to determine where the tank may respawn.
- getIsLoser() returns if the tank is a loser.
- takeDamage() subtracts the current health points of the tank by its damage value.

- heal() is a method used by the heal power up to restore some of the tank's health points without overflowing.
- changeSpeed() is a method used by the speed boost power up to change the speed of the tank as well as change the speed of which the animation plays.
- changeDelayBetweenShots() is a method used by barrage power up to change the cooldown / delay between shots.
- checkBorder() simply makes sure the tank is not able to go off screen.
- handleCollision() checks which GameObject the tank collides into to.
    - If the tank collides against a power up that call the power up's respective empower method and make sure the power up marks itself for destruction.
    - If the tank collides with the projectile and it is owned by a different tank, then the tank will take damage and mark the projectile for destruction.
    - If the tank collides with a wall or another tank, then we simply set the location of the tank to be outside of the wall / tank by using the smaller rectangle that is created when those two objects overlap each other.


(400,400)
(448,400)
(400,448)
(448,448)
Use this small intersecting rectangle to push tank away by width & height

        - The collision checking for wall and tank allows the tanks to push each other.
    - drawImage simply renders the tank's arrow image and hitboxes if hitbox mode is on.
    - checkOneShotOneKillMode() increases speed and damage to allow users to kill other tanks in one shot and when untoggled returns back to its initial values.

- **TankController:**
    - Constructor takes in a tank to control and 4 parameters that are used for key binds.
    - Implements KeyListener.
    - keyPressed() uses if statements to toggle what direction the tank moves in.
    - keyReleased() uses if statement to untoggled the direction the tank were moving in.

8.8.2 Projectile Package

The projectiles package contains classes that are extends projectiles. The current class that extends projectiles is bullets although you could also implement ideas such as fireballs using this package.

- **Projectile:**
  - Is an abstract class that implements Collidable.
  - Contains 2 protected fields Tank ownership and isDestroyed.
    - ownership is a reference to the tank that created the projectile to prevent the tank from damaging itself.
    - isDestroyed is used to check whether this projectile is to be removed from the GameWorld.
  - update() method moves the projectile every frame and checks if it goes past the world border and moves the hitbox of the projectile along with it.
  - checkBorder() is a private method similar to the tank and basically make sures to delete the projectile if it hits a world border.
  - getOwnership() returns a reference to the tank that owns the projectile.
  - setDestroyed() allows the user to destroy the projectile.
  - playSound is an abstract method that allows the user to create a custom sound for the projectile.
  - getIsDestroyed() returns isDestroyed to allow the projectile to be destroyed.
  - handleCollision() checks what the projectile should do when it encounters other GameObjects.
    - If it encounters a wall, it will destroy itself and play its sound.
      - If the wall was breakable, then also make sure to destroy the wall.
    - If the projectile encounters an enemy projectile, then that projectile destroys each other.
  - isCollidable() returns true since this projectile is collidable.
- **Bullet:**
  - Extends the Projectile abstract class
  - Its constructor is the same as the constructor of Projectile.
  - Overrides the playSound() method and uses the ownership field to get its collision sound.
  - drawImage() simply draws the image of the bullet and the hitbox if it needs to.

8.9 StationaryObjects Package

The stationary objects package contains classes related to GameObjects that are not allowed to be moved and may have special interactions when collided against such as breakable walls or power ups.

- **stationaryObject:**
  - An abstract class that extends the GameObject class and implements Collidable.
    - Contains one protected field called isDestroyed which is used to mark whether object is to be destroyed or not.
  - Contains 7 methods:
    - handleCollison() provided by Collidable interface.
    - isCollidable() provided by Collidable interace.

        - drawImage() simply draw the images although this should be an abstract method.
        - drawHitbox() is a method that draws the hitbox around stationary objects when hitboxes are turned on. The color for stationary objects is blue.
        - update() provided by GameObject.
        - getIsDestroyed() returns if the object is destroyed.
        - setDestroyed() sets the status of the stationaryObject.

8.9.1 Walls Package

The walls package contains code related to walls such as breakable and unbreakable walls.

- **Wall:**
    - An abstract class that extends StationaryObject and implements Collidable.
    - drawImage() simply draws the image it was given by its parameters and calls the StationaryObject's drawHitbox() method.
    - update() empty method provided by GameObject abstract class.
    - handleCollision() empty method provided by Collidable interface.
    - isCollidable() empty method provided by Collidable interface.
    - setDestroyed() is an abstract method meant to destroy objects. The reason this is abstract is because certain objects do not need setDestroyed() such as UnbreakableWall
    - getIsDestroyed() returns if the object is to be destroyed.
- **UnbreakableWall:**
    - Extends the Wall class.
    - Contains an empty setDestroyed() method because this object cannot actually be destroyed.
- **BreakableWall:**
    - Extends the Wall class.
    - Overrides the setDestroyed() method to set the object to be destroyed.
    - Contains a playSound() method to play a sound when it is destroyed.

8.9.2 Powerups Package

The power ups package contains code related to power ups.

- **Powerup:**
    - Extends StationaryObject and implements Collidable interface.
    - empower() is an abstract method that receives a tank and should give the tank a special power.
    - drawImage() simply draws the buffered image of the power up and the hitbox if hitbox mode is on.
    - update() is an empty method provided by the GameObject abstract class.
    - isCollidable() returns true since power ups are collidable.
    - getHitBox() returns the hitbox of the power up.
    - handleCollision() is an empty method provided by the Collidable interface.
    - playSound() is an abstract method that each sub-class of power up should implement as each power up should play a different sound.
- **Barrage:**

    - Extends the Powerup abstract class.
    - Overrides empower() and uses the tank's changeDelayBetweenShots() method to increase the rate of fire.
    - Overrides playSound() to play a custom sound when it is picked up.
- **Heal:**
    - Extends the Powerup abstract class.
    - Overrides empower() and uses the tank's heal method to restore health to the tank.
    - Overrides playSound() to play a custom sound when it is picked up.
- **SpeedBoost:**
    - Extends the Powerup abstract class.
    - Overrides empower() and uses the tank's changeSpeed() method to increase the speed at which the tank moves.
    - Overrides playSound() to play a custom sound when it is picked up.

9 Reflection

Overall, this was an enjoyable experience for me. I felt like I truly learned a lot more about how to write better scalable, maintainable, and cleaner code than I did before taking this course. I enjoyed reading up about what design patterns I could use to make my life easier like when I used the factory design pattern to encapsulate away the giant switch statement for dealing with game object creation. I also enjoyed writing the abstractions for the game objects such as moveable objects and stationary objects which was required for the project, but I also enjoyed making a MenuPanel abstraction so that all other menu panels in the menu package could extend and inherit from to reduce and reuse the same code that they all share. In the end, I am satisfied with the end product, and I know that if I had more time, I could make this project have even more features and better and cleaner code.

10 Project Conclusion

In conclusion, the project is working as intended as I was able to complete and fulfill the requirements given in the specifications such as:

1. Having a start screen allowing the player to start the game.
2. Having an end screen allowing the player to restart the game.
3. Having 2 playable tank characters
4. Having tanks that move forwards and backwards
5. Having tanks that rotate so they can move in all directions
6. Having a split screen
7. Having a mini map
8. Having a health bar for each player
9. Having a lives counter for each player
10. Having 3 power ups
11. Having unbreakable walls
12. Having breakable walls
13. Having bullets that collide with walls
14. Having bullets that collide with other tanks
15. Having a JAR stored in the correct folder on GitHub

16. Having correctly populated the README.md with the requested information

Besides the above requirements I also added my own goals that were intended to spice up my project which were:

1. Having a primitive map editor in the form of CSVs that the user could possibly add if they want to make a custom map
2. Having a map selection menu which allows players to select which map they want to play
3. Having a Help menu button
    a. Displays key binds for each player in game
    b. Display power ups and what they do
    c. Display cheat codes for debugging purposes
4. Having a restart button and a button to return to the main menu
5. Having a custom HUD for each player instead of just having health bar and lives count follow the tank around
6. Having bullets collide with enemy bullets
7. Having tanks being able to collide and push other tanks
8. Having customs sounds and music which stop when the game is done
9. Having custom walking animations for each player