CSC 413 Project Documentation

Fall 2021

Arielle Riray

917861209

CSC-413-

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

https://github.com/csc413-SFSU-Souza/csc413-secondgame-atrixatr

Table of Contents

2. Introduction

a. Project Overview

For this project, we had to build two games. The first one being a tank game, and the second being a game of our choice. I chose “Galactic Mail” as my second game, which was one of the games given in the list on iLearn by the instructor.

b. Introduction of the Tank game (general idea)

The objective of the game is to try to kill the other tank before they can kill you.

In this game, there are two players, meaning that two people can simultaneously play together on the same keyboard. The first player is the red tank, and the second player is the blue tank. Both tanks spawn on opposite sides of the map. You can see the map in the middle of the screen towards the bottom, where you can get a clear view where the other player is. You'll have to destroy some boxes in your way first to get to the other player, but the grey boxes are unbreakable.

There are also powerups the players can collect, where you can gain either an extra life, more defense, or more speed. Use these powerups in the corner and center of the map to help aid you in taking down your opponent.

Each tank both starts with 3 lives with 100 health. Each bullet deals 10 damage. Whoever reaches 0 lives first loses, while the other tank wins.

c. Introduction of the Second game (general Idea)

The player is on a spaceship on a mission to deliver mail to moons. In this game, you will see yourself start out on a moon. You will have to fly to other moons while avoiding the asteroids. However, the rocket doesn't have very good control. So once your rocket launches off the moon, it'll keep flying forward, and it's up to you to steer it out of harm's way while avoiding the asteroids in your path.

You're also a space mailman in a hurry, so the longer you linger on moons while trying to aim for the next moon, the lower your score becomes. So be prompt to get the best paycheck possible!

There are three levels, each level scaling in difficulty, from the moons and asteroids moving faster increasing in size, to your own rocket moving faster, making it harder to dodge the obstacles

3. Development environment.

The environment I used to build this project is Intellij by NetBeans. This is an IDE that specializes in JAVA. I also used the JDK 16.0 compiler. In addition, I also used Github for the repository for this project.

a. Version of Java Used

Version of Java Used: IntellJava(TM) SE Runtime Environment (build 1.8.0_261-b12)

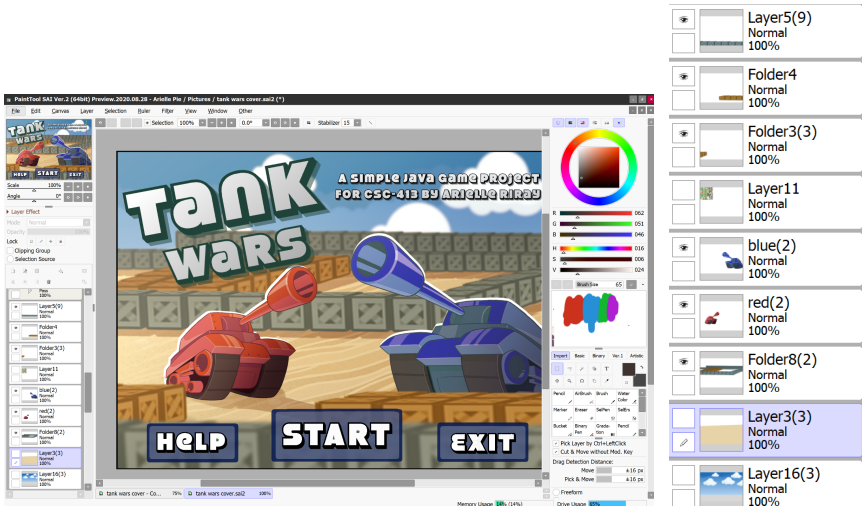
b. IDE Used

IDE used: openjdk-16.0.2 in Intellij by NetBean

c. Any special libraries used or special resources and where you got them from.

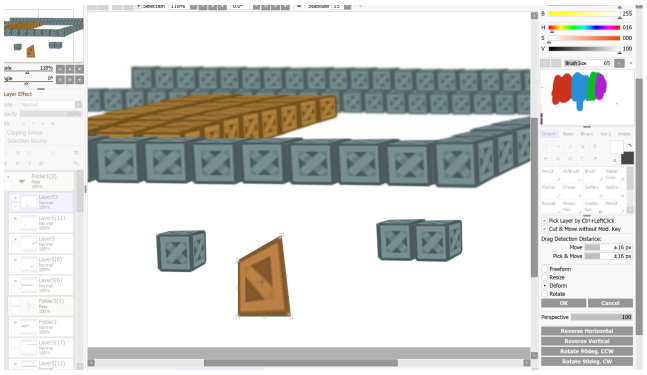
I got the resources for my tank game from this website:
https://opengameart.org/content/topdown-tanks

I had to do a lot of dimension and sprite adjusting to have the parts fit in my game.

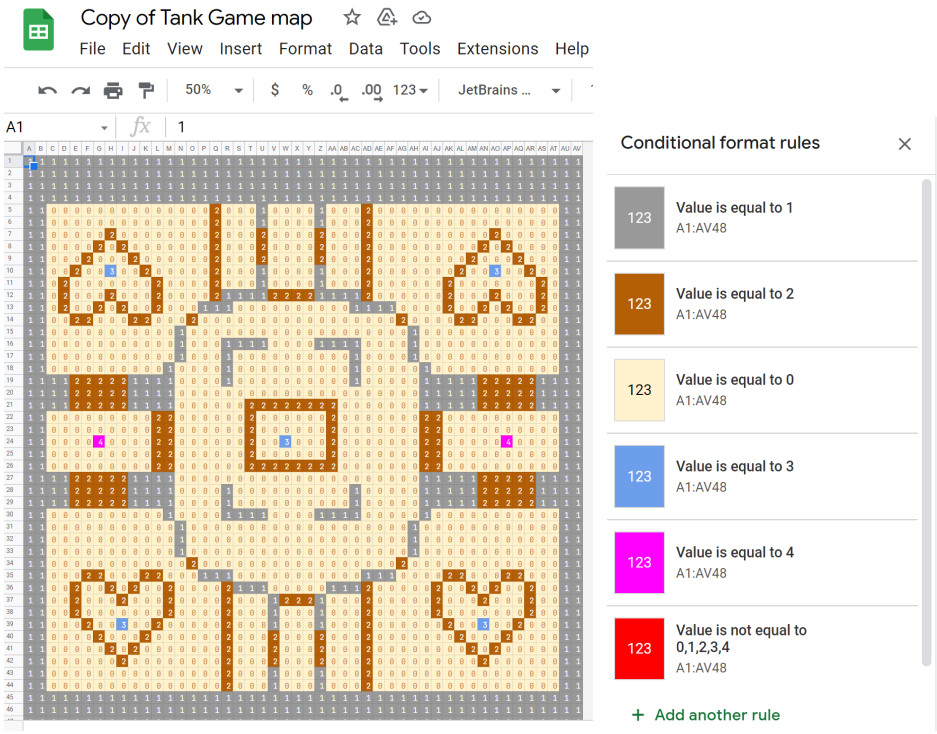


Also, here's some pictures of me working on creating the main menu screen for the tank game! As for the title screen and Help screen, I actually put it together using the assets as textures. you can see by the layers in my program.

Below is a little bonus of how I created the boxes from the texture, and stacked them in a three dimensional way to create depth.



I also had my little brother create the Game Map for me! Through using a conditional google sheets format where the number backgrounds would change based on what number is in the array, this is what I let him work with. After he made the map, I exported it back out as a CSV file where I can place it back into my code ready for use!



I also referenced a lot of code from the instructor's videos, and the example code TRE provided to us.

4. How to build or import your game in the IDE you used.

a. Note saying things like hit the play button and/or click import project is not enough. You need to explain how to import and/or build the game.

1. Download or Clone repo to your computer
2. Open IntelliJ
3. Go to "Open Project" in Intellij, and select "csc413-tankgame-atrixatr"
4. and Go to the "Project Structure" in the File menu. Then do the following:
    - Set this repository "csc413-tankgame-atrixatr" as the content root (You can do this if the project wasn't already open)
    - Mark the "src" folder as a source folder
    - Mark the "resources" folder as a resource folder

Then setup the JDK to openjdk-16.0.2 in the Project Structure or by clicking "Define JDK" in the top right if indicating After the IDE scans the code and indicates its ready to run, you can try building. You can click the green bug button at the top, or navigate to the "Build" tab at the top, click on it, and then click "Build Project"

b. List what Commands that were run when building the JAR. Or Steps taken to build jar.

To build the jar file, I navigated back to “Project Structure” in the File tab. Then I click on it, and move my mouse down to the “Build Artifacts” tab. Click on it, go to “jar”.

Then choose “build module with dependencies”. A screen will pop up, and “Gameworld.java” file as the main file to run the jar. Click apply and OK. After closing out the screen, navigate mouse to the Build tab at the top.

Then click Build artifacts and select “jar”. After that, the IDE will take a few seconds to generate the jar in the out/production/artifacts folder.

c. List commands needed to run the built jar

To run the jar from the ide, you can navigate your mouse to the `out/production/artifacts folder.’ Then right click on it, and hit ‘run’.

5. How to run your game. As well as the rules and controls of the game.

You can click the green play button at top, or directly to the class "LaunchGame" in the src/Tankwars folder and click on the green play button shown next to the code.

Or you can navigate to the Jar folder, open it, right click on the "csc413-tankgame-atrix.atr.jar", and run it

Tank game:

Shoot your opponent's tank with bullets. The more they get hit, the lower their health goes down. When their health reaches 0, they lose a life. Take all the lives of your opponent and you win. There are also powerups scattered throughout the map in which you can collect.

	Player 1	Player 2
Forward	W	UP ARROW
Backward	S	DOWN ARROW
Rotate left	A	LEFT ARROW
Rotate Right	D	RIGHT ARROW
Shoot	SPACE	ENTER

Galactic Mail:

You need to land on all the moons on the screen to deliver mail. The longer you stay on those moons, the lower your pay goes down. Also, your ship has limited controls, and you're only only to rotate through space while your ship surges forward on its own.

Try to finish delivering mail to all the moons in the three levels without getting hit by asteroids.

Controls: A-rotate left, D-rotate right, SPACEBAR- Launch off moon

6. Assumptions Made when designing and implementing both games.

I'll compile the assumptions into a list rather than paragraphs:

-I thought that I'd be able to create my own second game. Unfortunately, due to a few reasons, I didn't finish making it, and I ended up going with one of the games given out by choice from the Instructor. (I'll elaborate upon this in the reflection.)

-I had a habit of trying to call functions or integers in another class, but I forget to instantiate them. This created a lot of "cannot reference from a non-static context" errors

- I thought that bullets already had their own delay when the shoot button was pressed down. However, if held down long enough, the function spawns bullets every tick. I had to implement a millisecond checker function to stop the bullets from spawning if the player holds down the shoot button.

-I thought that galactic mail was the easier game to make out of the list since I thought it was the most similar to Tank Game. Turns out that according to the professor in a discord mmessage, Super Rainbow Reef was the most similar.

-I thought that I could initialize the whole game at once. However, this caused issues, because the user can play the game in the menu screen, with the game happening in the background.I learned my lesson in the second game when I accidentally heard the death effect play despite still being in the main menu screen. I only realized this after I turned on Tank Game, so Tank Game can actually be played from the menu, but Galactic Mail cannot.

-I thought that I could try to have the setLevel() function in Galactic Game in a separate class like I did with Tank Game. However, the game became more laggy, sharing the Game Objects List between setlevel and Gameworld, so I ended up having the init, setLevel, draw, and collision functions all in the same driver file. I don't know if this is poor practice, but it made coding easier with variables all being in the same class file.

I thought that I could use the bullet class in place of the asteroid. I ended up changing my abstract/template class structure in the second game, so the parts of the bullet class actually became shared between my ship and asteroid objects

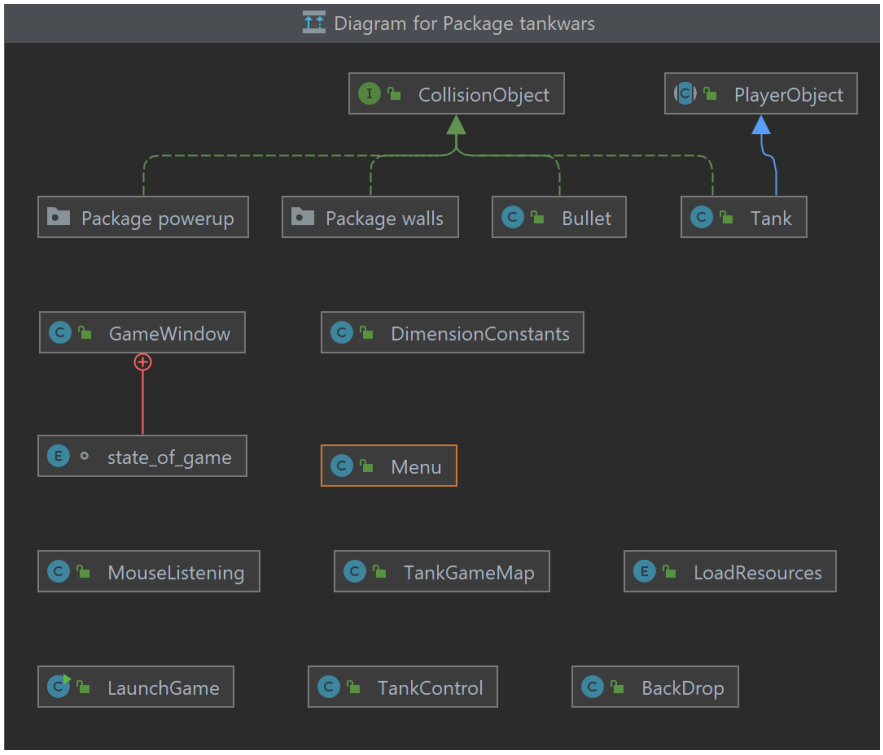
-I thought that my collision hit box was a bit too much in my first game, so I shortened it down in my second game. Not sure if this is detrimental, but the game still functions.

-I thought that I could make all the game objects in the Tank Game unified under the same abstract class. This ended up being too difficult for me to successfully implement, so now there's two abstract classes for each respective package, of the Wall and Powerups class. There is most definitely a way to do this under one class, but I found it easier to have its own classes. Still, both abstract classes were under the Collision template.

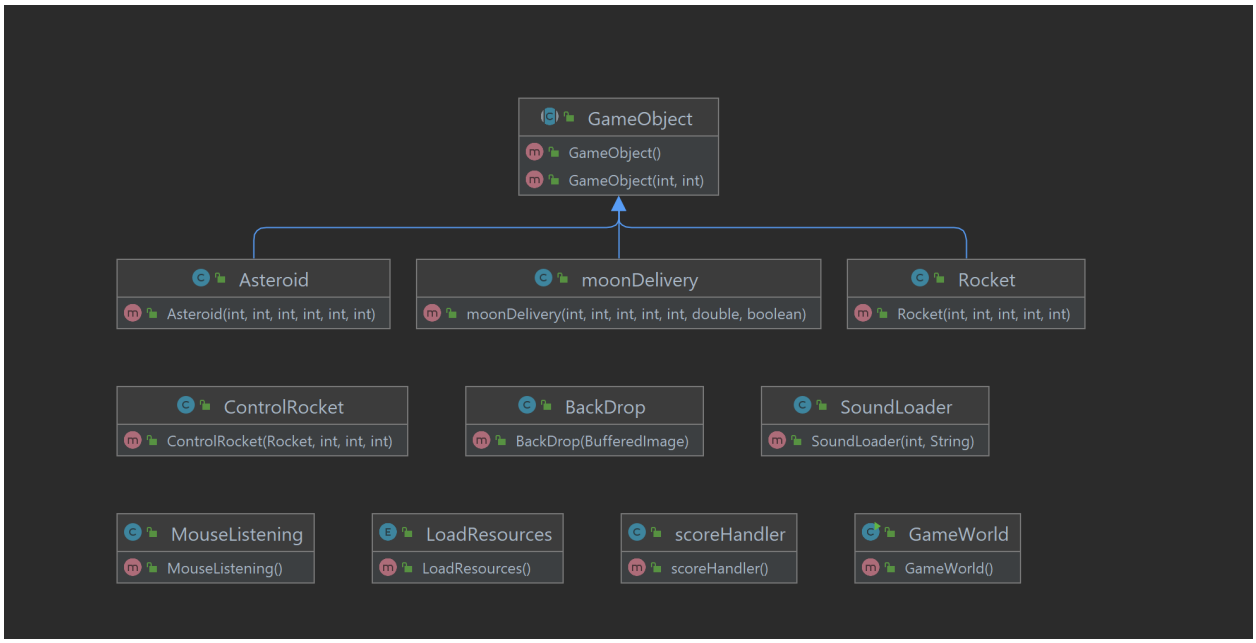
-I ended up trying to make only one abstract class for all the game objects in the Second Game. I think in the process, I ended up having more disorganized code.

-I think the utility classes are fine to stand on their own without a package

7. Tank Game Class Diagram



8. Second Game Class Diagram



9. Class descriptions of all classes shared among both Games

Main Classes

Overview:

Both games share a majority of its classes. The main idea is the collision class being either a template, or an abstract class shared amongst the game objects that would need a rectangle for it's hitbox. Another similarity in class structure are game objects sharing the majority of classes or being in it's own specific packages according to their types.

Also, in the main class, an init function would be called, initializing game objects, followed by a level creating function, a draw function to draw all the components to map, with an update function all in a while loop to keep the game continuously running until it's over. Both games also use game states to determine which stage of the game they're in.

I will be labeling the classes in the next two sections in the following forms/colors to hopefully make the identification process easier to read for the viewer.

MAIN CLASSES ARE IN BOLD
subclasses are italicized

Shared Classes between both Games
Tank Game Specific Classes
Galactic Mail Specific Classes

MAIN CLASSES THAT ARE SHARED

PlayerObject/GameObject

Despite having different names, both of them have very similar functions. they store the setters and getters of all the movement variables to be shared amongst the objects that'd be moving.

BackDrop

This class loads in the background of both games.

private void loadImage(BufferedImage img) {this.background = img;}
public BackDrop(BufferedImage img) { loadImage(img); }

These classes load and set the image for the BackDrop drawImage class to use

void drawImage(Graphics g) {

This class loads in the background seen in the game. It's the first draw function to keep updating, because it needs to be behind the rest of the assets. The reason why this particular asset needs to have it's own class, is because it needs to be repeatedly buffered in a specific way defined in its class, to cover up the drawn objects of the other assets.

What this means is that when moving game objects are repeatedly drawn on the screen, they can be left to the player to see. This is quite ugly to look at. With this background class, it helps cover up the tracks made behind other drawn objects.

In the tank game, this function is called within a FOR LOOP, since the actual background file is quite small and would need to be drawn repeatedly to cover the entire map. There is no FOR LOOP in the BackDrop class of Galactic Mail

GameWindow

This class does most of the heavy lifting within both games. Both are slightly different however and will be elaborated in the next sections.

main()

This function is what allows the whole game to execute and run.

init()
This function is short for 'initialize', because it initializes all the necessary game components and objects, such as the jFrame window, setting it's attributes, initializing the background, creation of the map, the player, and the rest of the game objects.

public void paintComponent(Graphics g) or drawimage()

This class is a must if you want to draw images in an application. As for video games, that's usually the case. This class allows the user to see the images and assets of game objects. It does this by drawing in Buffered images called in from the LoadResources class. It can also draw in rectangles, outlines, shapes, and text. This is an all around handy tool for programmers to know if they want to create an appealing friendly user interface.

LoadResources

public enum LoadResources{}

This lists out all the constants' names assigned to its specific images.

public BufferedImage getImage()

This function returns the image that was loaded in this class. Another class can call this if it wants to use a specific image, they just need to have a Bufferedimage data type initialized to be able to use this function.

public static void init() {}

This function uses the following code of:
LoadResources.name.image =
ImageIO.read(LoadResources.class.getClassLoader().getResource("filename"));
This code loads in all the resources using this specific statement. It also loads it all in using a try catch statement in case of null images/errors, along with being in a nice list of constants of an Enum format.

This class is shared between both games, the only difference being that it loads the resources specific to its game. For example, this class loads in 14 assets for tank game, while the galactic mail has less to load.

MouseListening

This class is also the same between both games. To make things easier, I made both games share the same screen resolution, so the dimensions for the mouse reader are consistent between both. The point of this class is to allow the user to interact with their screen with mouse input, from where the mouse is clicked in terms of coordinates, or to whether the right/left mouse button is toggled.

Tank/Rocket

This class shares very similar attributes with each other. I'll delve more about this in the following sections.

TankControl/RocketControl

This class holds Key listener Functions such as
public void keyTyped(), public void keyPressed(), public void keyReleased()

These classes share almost the exact same code, and have the function of checking when a specific key is pressed down, so it can set the boolean assigned to that key, which other classes can utilize.
The only difference with the two games, is that Galactic Mail is missing the up and down input keys, because the game doesn't require them. Unlike tank games, where it does use those keys.

SHARED SUBCLASSES

TankGameMap/setMap

Despite having different names, the way these functions operate is very similar. Using a 2D array, this function loads in the map components through multiple for loops.
How this works, is that I fill in an int array with specific dimensions, ranging from the values of 0,1,2. The 0 represents empty space, where nothing is to be loaded in. Then the 1 and 2 represent actual in game objects. These vary which I'll explain in the later sections.

How this function works is that it first initializes a row variable, and a map variable to 0. These there will be a for loop in charge of cycling through a column. Right after is another for loop which will increase in value after all the values in the column are cycled through. The map and row variables are incremented up by one, since these variables are used to fetch the values from the array. This process repeats until the whole map is loaded in. Whenever there is a 1 or a 2, an object will be placed in that particular spot. This makes map building very streamlined.

Also, the maps are built in google sheets with conditional formatting to make the building process easier. I actually had my little brother build the tank game map for me, and I think he did a wonderful job. Especially with the distribution of the powerups, with the most desirable one being in the middle.

private void rotateRight()

private void rotateLeft()

private void moveForwards()

private void moveBackwards()

These four classes are the same in both games. The controls are almost the exact same in both games and utilize the same movement akin to most top-down 2D game.s The rotate classes adds or subtracts the angle variable to the rotate speed variable.

Then they move forwards/backwards classes add the (x+vx) and (y+vy) to create the illusion of velocity, of the object actually moving. The vx and vy variables can also work with rotational movement, with the (int) Math.round(Speed * Math.cos(Math.toRadians(angle))) function, allowing more movement than just 4 ways.

private void checkBorder()

This class checks the screen borders whenever an object goes out of bounds. In the tank game, it despawns the bullets, while in Galactic Mail, it sends the object to the opposite side of the map.

public abstract void collide()

This class checks for whenever two objects' rectangles/hitboxes collide, a very important factor for video games to work.

Constructor: public GameObjectName
Every game object class needs a constructor so that their variables can be called and set from other classes.

public Rectangle getRectangle() / getRect()

This class returns the hitbox of the object it's referenced to.

The Setters and Getters

These include functions such as public setVariable(type variablename) / type get() return variable

These functions would apply to the variables: x,y,vx, vy, angle, r(speed, and booleans. These are used for the program to set and get the values of these variables according to the constructor object they're associated to.

public void update()

This function runs usually every tick, where every necessary aspect of the game is updated, whether it's the movement of objects, hitboxes, draw functions, checking for game states, and so on.

Overall, these are the main shared classes and subclasses in both games.

10. Class Descriptions of classes specific to Tank Game

The object classes in the Powerup and Wall packages

Powerup

public abstract class PowerUp()

This class is an abstract class meant to hold the classes that'd be shared amongst all the powerups.

ExtraLife()

This function is meant to give the tank that collides into it an extra life. This is called in the TankGameMap when colliding. By calling a function in the tank class, it sets the health of a tank to 100, whilst granting them an extra life.

MoreDefense()

This function is meant to give the tank that collides into it more defense, and is called in the TankGameMap when colliding. By calling a function in the tank class, it brings down the damage of the other tank by a few points, which lets the tank with the powerup survive a few more hits.

SpeedBoost()
This function is meant to give the tank that collides into it more speed, and is called in the TankGameMap when colliding. By calling a function in the tank class, it increases the speed variable of the tank by a few points, granting the tank more mobility.

public abstract class Wall`

This class is an abstract class meant to hold the classes that'd be shared amongst all the Wall objects, which there are only two:

BreakableWall()

This class has a collide check class that checks when a bullet rectangle has intersected with its own. When this collision is true, this wall would disappear, hence why it's breakable.Also, it stops the player from moving into the wall, along with stopping bullets.

UnbreakableWall()

This class has an empty collision check class for bullets so it doesn't break. Also, it stops the player from moving into the wall, along with stopping bullets.

Overview of Wall and Powerup abstract classes

In both of these abstract classes, along with their associated classes under them, all of them actually share the same subclasses, which I will describe in more detail below:

public Rectangle getRect()

This empty class implements the CollisionObject template, which fetches the hitbox of the object it's assigned to in the child classes.

public void drawImage(Graphics2D drawImage){ }

This class is in charge of drawing the image associated with its object.

public abstract boolean impact();

This acts as a boolean to determine when a bullet impact has occurred

public void CollideCheck(CollisionObject CO) { }

This class detects when an object has collided with another object by taking in the class' constructor, or rectangular hitbox as an argument and comparing it with another object's rectangle.

public interface CollisionObject {}

This is a template class used in Tank Game. This was later removed in the second game due to it's redundancy in the abstract Object class

Menu()

This class is pretty much the same function between both games. It utilized drawString functions to help me keep track of where to create the buttons for the menu in my games. It is currently not in use by either class as it was meant as a support class for the development process.

Bullet()

This class deals with the bullets shot from the tank. A lot of the code was created with help thanks to the professor's videos.

public boolean impact()

This is a boolean that returns whenever a bullet has hit something. Commonly used with the CollisionObject interface class.

void setOwner(String owner)

This class helped me keep track of bullets shot from a specific tank for debugging purposes.

public static void setBSpeed()
This was made more for fun where I tested out how bullets would be at different speeds. With a very low delay and high speed, the bullets would act almost like a laser. I didn't implement this however, due to complications.

public Bullet()
This is a constructor to set the values of the bullet object, so other classes can use it. The vx/vy is also shared with the tank class in how the tank moves. This was also used in code provided by the professor in his TRE code.

private void checkBorder()

This class checks if a bullet reaches out of bounds, such as past the map edge. When the bullet is within range of the border, it stays spawned as true. However, when it goes beyond the border, the bullet despawns. This ended up not being used as much towards the end after implementing a border of unbreakable walls, but this function came very much in handy for the second game.

public void drawImage()

This function draws the bullets, and takes into consideration the angles it's shot from the tank. It also draws an explosion image whenever the bullet collides with something using the collided boolean in an if statement.

public void update()
This function runs every tick, where the bullet object will keep heading towards its specific direction until it collides with something. The checkBorder function and setLocation function are always run with this function as well.

public void CollideCheck()
This class implements the CollisionCheck template to check if it's rectangle intersects with another object's rectangle, which sets the collided boolean as true.

public Rectangle getRect()
This class returns the bullet's hitbox, or rather rectangle dimensions and placement.

TankGameMap()

I've already described in detail before how this function works in the section above. However, there are some parts to this class that's not shared with the Galactic Mail game. This class also spawns where the power-ups in Tank Game would be. Galactic Mail didn't appear to be a game that had power ups, so I didn't include them in my second game. The powerups would be added through a " powerUps.add(new powerupName(x, y)); function, the name depending on which powerup I want summoned.

This class also has a handleCollision section. It uses the CollisionObject template to check when a wall or powerup has been touched by a bullet or a tank. If it's a bullet, it makes the breakable wall break, but not the unbreakable one. As for powerups, they're not affected by bullets, but they will disappear and call the powerup when touched by a tank.

Finally, there is the draw image function. Using a for loop, it spawns in the walls according to the number read from the int Array. It also does the same with powerups.

Overall, this is a solid class meant for building most of the map components, and handling the collision of said components.

Tank() (Classes found only in Tank and not Rocket)

addLife() getLives() powerHealth() removeLife()

These classes control the health and lives of the tank. The tank starts off with 100 health. With each bullet shot, the health goes down. When it reaches 0, a life is taken off and health is restored back to 100 if a life can be subtracted before hitting 0. The extraLife powerup calls one of these classes to add an extra life to the Lives counter.

shootBullet()

Summons a bullet object from the bullet() class, where a bullet is shot from the tank, and the object is added to the bullet array

GameWindow()

protected void init()

This function initializes all necessary game instances, including the jFrame, tanks, mouse/key listeners, starting the LoadResources function, and loading in the buffered images to be called in draw functions.

private int getXcoord()
private int getYcoord()

Both of these methods get the coordinates of the tanks, and keep their coordinates in the center of the screen with divide/2 functions to give a feeling of a camera following the tank until a border is reached.

public void paintComponent()

This uses game states to determine with components to paint to screen. The menu and help states having their own respective images shown on screen before the game starts. When the game starts, the tanks, blocks, powerups, and background are all continuously drawn. This class uses the getX/Ycoord() classes to draw out the split screen. A minimap is also drawn at the

bottom, with player stats on the top. If a player dies, a win screen would be drawn for the winning player.

11. Class Descriptions of classes specific to Second Game

SoundLoader()

This class was actually borrowed from the airstrike game. I didn't implement sound in my tank, so I wanted to try it out on my second game.
I used Audacity to splice up a song with a chill, retro-synthwave feel since it fit the theme of space to me. The song has this build up section that I wanted to play on loop in the menu, so I used Audacity to chop up and modify the song to my liking.
When the game starts, the song jumps to the chorus. This class also has if statements to determine if a song would be played on a continuous loop or just once, such as with sounds like an explosion, launch, or 'bonus' sound.

GameWorld()
While this was a shared class with a tank game with the same name, there are vast amounts of differences in it.

main()

This function no longer consists of just an init() function like in Tank Game. Now it has a series of if statements determining the game's state, which can be changed from the MouseListening class. The reason why I implemented this change, is because the game would actually still run in the background when all init parts are initialized. So I split init up in two parts, so only the necessary parts to run in the menu would run, with less risk of the player starting the game in the wrong state.
This function also keeps track of what level it is, what game state it is, and calls the collision class if a level is in session and the program is in a game state. It will also go up a level when a stage is cleared, and increase the speed of some objects.
If the application is no longer in a game state, it will clear the game objects from the list and call the repaint() function to paint the lose/win screen.

MenuInit()

This initial function consists of all every init function from tank game, except with the keyboard listeners not initialized yet. That way the player doesn't start too early. It still initializes components such as the background, player, mouse listener, jframe, and images. Also, it begins the menu song loop.

Init()

This function when called after game state = game, then initializes the keyboard listeners needed to play the game, so the player can start the game when they're supposed to. The song also changes as well.

setLevel()

This worked very similarly to the TankGameMap function, where it utilizes 2D array reading to load a map. The difference however, is that there are now 3 different int arrays to read from, each representing a level. The 0s represent empty space, the 1s summon asteroid objects, and the 2s summon moon objects. These 2s go through a conditional depending on which level they're summoned on. on the first level, they have no speed value, so they don't move. On the following levels, they have movement speed, and it increases with each level.
Also, with each moon and asteroid loaded, they are added into a game objects list, which other functions can utilize.

CheckCollision()

This collision conditional statement is a lot more shortened from the tank game, and checks between if the rocket has collided into a moon or an asteroid. Whichever object is detected colliding into, its respective function is called, whether its death from a colliding asteroid, or having the ship's X/Y values share the moon's upon collision.
After the object is collided, it gets removed from the list. This comes in handy with the moonChecker variable, which signals to the main() class when it's time to move onto the next level.

paintComponent()

This function paints the components to the screen. It also uses conditionals to paint images according to the state the game is in.

GameObjects()

This is an abstract class to hold the functions for Asteroid(), MoonDelivery(), and Rocket().

This time I tried making the player and the game objects under the same abstract class, since they now all utilize movement and hitboxes.

Asteroid()
MoonDelivery()

These classes don't hold different classes from Tank Game. But now they have bigger constructors since they are able to move and need the speed/angle variables.

Rocket()

I removed the bullet class related methods from the (Tank->renamed->Rocket) class, because the ship doesn't shoot bullets in this game.

Almost all classes are pretty much the same, with the exception of its collide() class.

collide()

There are a few conditionals in this class, each with a different outcome depending on which object's rectangle meets the ship's rectangle. This class was a very finicky class to work with, as I couldn't figure out how to resolve a few bugs, even though this class was the source of the bugs.

If the rocket hits an asteroid, it'd play an explosion sound, and set the game state to 'over', where a loser screen is drawn out in the ScoreHandler() class. I also put a millisecond delay checker so the explosion sound doesn't play more than once.

When the rocket hits a moon, it'd first check if the game is still in game state. If it is, then it'll play a landing sound, which also has a millisecond delay checker, along with a boolean that'd only switch when the rocket is fully off the moon.
An "onMoon" boolean would switch on, which would make the rocket immune to the asteroid in the time being. Then the x, y, vx, vy variables would be shared from the moon to the rocket, to simulate as if the rocket actually landed on the moon and is moving along with it. During this time, a scoreHandler() method is called where points would go down by the milliseconds in the game.

When the player decides to take off, a launch sound will play, and this is detected by the "this.LaunchPressed)" condition, where the player presses the spacebar. The rocket gets launched, and thescore stops counting down. The onMoon boolean is switched, making the player susceptible to asteroids again.

ScoreHandler()

Since a big part of this game is the way their scoring system works, I decided to make this its own class, where all score related functions would be at.

setPoints()

This function adds points whenever a player lands on a moon

public long lostPoints = System.currentTimeMillis();

Not a function, but a useful variable to count the score down when the player is on a moon

drawImage()

This draws the current score/money on the screen while the game is in game state. When the game enters a win or lose state, it'd draw text depending on the game state.

Overall, these are the main differences my Galactic Mail has from my Tank Game in it's specific classes.

12. Self-reflection on Development process during the term project

This project made me realize I had so much more to learn. It's as if the more I learn, the more questions I end up having. Overall, despite all the frustrations and long hours spent trying to figure out a simple issue in my code, the payoff was downright satisfying. I didn't get to achieve all the things I wanted to get down for my second game

One of the biggest obstacles I've faced during this project was the issues regarding my device. I use a tablet/laptop hybrid, and throughout the years, I've noticed that it's battery life seems to get weaker and weaker with every usage. Then one day, late November, the battery in my laptop suddenly exploded. Not only did my tablet contain a lot of sentimental photos, but it contained the project files to all my classes. Because this happened so close to the tank game deadline, I almost gave up. Almost to the point I was considering dropping the class, because I felt there was no way I could finish the game in time.

Luckily,very thankfully, my professor has allowed me to have an extension after explaining the situation to him. It was still a hassle trying to use other people's devices to get this project done however. Still, through it all, I managed to finish the game, and I'm kind of proud how the tank game turned out.

I took liberty upon viewing other people's tank game submissions on youtube afterwards, and while it's true there are many CSC-413 tank games leagues far better and more feature fiddled than mine, I'm still proud of what I was able to do. Even without sound, or animation, just the fact that I learned how to build my very own first game from scratch without a game engine, is something that makes me feel pretty happy with myself.

As for the second game, that is a bit of a different story. I did have considerably less time to work on it, but the issue was the choice of the second game I wanted. Initially, I wanted a shooter game where the player has to try to survive waves of monsters.
However, I did this with the intent of making sprites and assets in dedication to another game I play, which is Hypixel Skyblock. It's a game I've grinded thousands of hours on, I love playing it and I enjoy the community. However, midway through making my second game in dedication to Hypixel Skyblock, I suddenly got permanently banned from that game with thousands of hours of progress wiped with no explanation why.

This crushed me. I could no longer find the motivation to make this specific second game I've started, because I started it with the intent to share my game with the Hypixel Skyblock community. It hurt everytime I tried to work on this themed second game, and eventually, I just scraped the whole thing.

With the deadline approaching, I decided to take on one of the default games offered from iLearn. I chose Galactic Mail since I thought it’s the most similar to Tank game.

Because I started on this second game so late, I feel it came out half baked. This is my fault of course, and I could see many ways I could improve upon it. Still, I’m glad I was still able to get it done, but I think I’m still gonna spend my off time trying to improve upon it.

Overall, this project was an emotionally heavy, yet valuable learning experience. This project ESPECIALLY helped me understand ways classes can interact with each other, and how templates/abstract classes worked. I really enjoyed the stuff I learned from this class, and it makes me want to try to attempt to make more games. They're not gonna be as polished or good, but I like to learn from my mistakes with each new coding project I take.

13. Project Conclusion.

Overall, I'm satisfied with how my first game came out, but not quite the second game.

There were these two pesky bugs in my second game that I couldn't figure out how to fix.

For example, about half the time, the ship doesn't stay on the moon when the moon hits the screen border, and actually goes off the moon by itself. Quite funnily, this makes the player actually invincible to the asteroids, since the Launchpress() command has the function to make the ship collide able again.

The other issue is that when the ship is on a moon, and another moving moon collides with the moon harboring the ship, the ship suddenly jumps to the other moon.

I've figured out the lines of code causing this issue, but I think it;s more of a fundamental flaw of how I implemented the whole game, which makes it difficult for me to try to fix the bugs. I have a lot more to learn

Also, of course, I'd like to learn in the future how to successfully implement animation into games, and work with more assets. However, functionality is more important than those auxiliary-like items, and is what I need to focus on improving.

I understand the many flaws in my games, and that I'd probably get dinged for them. Still, overall, I'm happy with what I learned from this class, and am looking forward to learning more.

However, I would have to brush up on some of the concepts of Object Oriented programming again before moving forward to harder endeavors. Perhaps in the future I'll continue polishing up these current games and try to improve their functionality.