CSC 413 Project Documentation

Fall 2021

Fara Yan

922290860

CSC 0413-02

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

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

Table of Contents

1 Introduction 4

1.1 Project Overview 4

1.2 Introduction of Tank Game 4

1.3 Introduction of Second Game: Sundays at Dim Sum (Rainbow Reef game) 5

2. Development Environment 5

3. How to Build/Import Project 6

4. How to Run Project 6

5. Assumption Made 6

6 Implementation Discussion 7

6.1 Tank Game Class Diagram 7

6.2 Sundays at Dim Sum (Rainbow Reef Game) Class Diagram 7

7 Class Descriptions of Classes Shared Between Two Games 7

7.1 GameObject Class 7

7.2 Moveable Class 8

7.3 TankControl/PlayerControl Class 9

7.4 Tile/Wall Abstract Class 9

7.5 UnbreakTile/UnBreakWall Class 10

7.6 HealthTile/HealthPowerWall Class 10

7.7 SpeedPowerWall/SlowSpeedTile Class 11

7.8 Collidable Interface 11

7.9 GameConstants Class 11

7.10 MapLoader Class 12

7.11 Resource Class 12

7.12 EndGamePanel Class 13

7.13 StartMenuPanel Class 13

7.14 LivesBar Class 14

7.15 Scoreboard/Player Class 14

8 Class Descriptions of Classes Specific to Tank Game 15

8.1 Bullet Class 15

8.2 Tank Class 15

8.3 SizePowerWall Class 16

8.4 HealthCount Class 17

8.5 TankExample Class 17
9 Class Descriptions of Classes Specific to Second Game 19
9.1 Ball Class 19
9.2 Player Class 19
9.3 ReefDisplay Class 19
9.4 Sound Class 20
10 Project Reflection 21
11 Project Conclusion/Results 21

1 Introduction

1.1 Project Overview

This project includes two games that give users the ability to move a player to win or lose a game. This project opens a separate window the game is run on, and keys on the keyboard that give users directional control (left right, and/or up down). In both projects there are the presence of collisions, where two objects will display some reaction when they encounter each other. This will be the premise of how both games are played out, where users have control of two things: one with complete control, and one to control for a short duration of time. The first game will have a tank and bullet respectively, and the second game will have a platform and ball respectively. Both games also have obstacles players can encounter: blocks. Blocks are stationary objects that can be either unbreakable or breakable when collided with the right object. They will then disappear and give some benefit to the player, whether it is a special power or increase in score. Players are also given a limit to end the game; players have three lives before a game ends. In total, users are presented with a starting view, game view, and ending view.

1.2 Introduction of Tank Game

Aboard Sunset Wonderland: The Tank game is a two-player game where each player is in control of a Tank. This game takes on a fun theme of Sunset Wonderland, a dreamy pastel environment where players are bunnies travelling through a magical unknown.

General Rules/Goal: Each player has their own half-screen view to navigate around a map, and a mini map is also shown to guide players through their environment. Players move around using directional keys (player one controls WASD, player two controls arrow keys) and shoot bullets in their direction using either the enter or space key. The overarching purpose of the game is to shoot the other player with bullets until they lose their three lives. Each life contains 20 health points, and one bullet diminishes 10 health points from a player. Because bunnies are quite weak, these health points do not come back.

Blocks: Along the way, they can also shoot their bullet towards blocks in their environment. Players are not able to travel through these blocks, nor travel through another player. When they shoot blocks, the block hit and bullet used both disappear. A player never runs out of bullets however, but the number of blocks is finite and does not get regenerated. Players can use blocks to their advantage in two ways. One is to shoot blocks to carve open a path in which they can shoot towards a player.

Special effect blocks: Another advantage players can gain is to hit one of three special effect blocks: a Restorative Everpink Tree, Super Swirl Tree, or Geode block.

- **Restorative Everpink Tree:** Hitting bullets towards a Restorative Everpink Tree fully restores a player's health to three lives and 200 health points, as bunnies eat the magically healing Everpink leaves.
- **Super Swirl Tree:** Hitting bullets towards a Super Swirl Tree increases the size and strength of a player's next three swirling bullets. This will give a bullet twice as much damaging effect, and a greater chance of hitting the object they target, as the size is bigger.
- **Geode Block:** Hitting bullets towards a Geode Block increases the player speed by 2.5x. speed to travel around Wonderland with greater agility. (Bunnies are naturally defensive animals, but the

armor they gain from a geode block makes them move with confidence! In this way, they move about with faster speed, instead of being overly cautious of their surroundings. A player's speed goes back to normal when the other player throws a swirling bullet at them, depleting their confidence to take on the world.)

1.3 Introduction of Second Game: Sundays at Dim Sum (Rainbow Reef game)

Let's Eat! Welcome to Sundays at Dim Sum: Sundays at Dim Sum is a remix off the Rainbow Reef game which is a single-player game where players control a platform to bounce a ball with. In this game, the platform is a small dish of food and the ball is a food item from that dish. This theme originated from my love for going to dim sum with my family, which is a meal where we order a variety of small Chinese dishes to all share.

General Premise/Goal: The overarching goal of the game is to collide the food item with all of the bosses in all three levels. Players can do this by moving their platform left and right to collide with the ball in an attempt to reach a boss. In this case, the bosses are members of my family, who I enjoy dim sum with. If the food item falls out of bounds, a player will lose one of their three lives. Once a player loses all of their lives or completes all three levels, the game ends.

Ball characteristics: Ball bounces are freely generated, so there is no way to completely determine the direction of the food item. However, if the bounces occur on the left side of the platform, food items will be sent in a general left direction. This pattern follows for bounces on the right side of the platform sending food items in a general right direction. With each bounce of the food item on the platform, food is delivered with greater urgency, in an attempt to reach family members before it gets cold from being thrown around so much. Thus, the speed of the food item will increase each time it collides with the platform until it reaches a maximum speed it can travel for safety reasons across a dinner table (this is to prevent outbreaks of food fights; dim sum is meant to be a relaxing experience!).

Blocks: The game will also contain a series of blocks that block the food item from being reached by a family member. These blocks can be broken after a specific number of collisions from the player, which varies from one to five collisions. Breaking each blocks also increases player score by 10, and can earn them advantages if they land on special blocks.

- **Extra Score:** Colliding with this block will give players a 50 point score increase, as opposed to the normal 10 point increase.
- **Extra Life:** Colliding with this block gives players an extra life, which can be advantageous in keeping them in the game.
- **Slow Down:** Colliding with this block slows down the food item speed. This helps the player have greater preparation time to hit the food item on the platform.

2. Development Environment

The version of Java I used is java version "16.0.1". The IDE I used was IntelliJ IDEA. I used the Java Swing package, a set of components to create window-based applications. I specifically used classes such as

JFrame, JPanel, JButton, and JLabel to organize and build the layout of my game. These resources are from the Java Foundation Classes, a framework to build Java graphical user interfaces.

3. How to Build/Import Project

To import my project, one must first download my project and install and open up IntelliJ IDEA. Once the application opens up, one must then press "Open" in the top right corner, then click on my project by searching for it in their list of files. Once the project opens, a screen with a directory of the files on the left should show, as well as code on the right portion of the screen. One should click on the csc413-tankgame-faraayan folder or csc413-secondgame-faraayan folder, depending on which game it is they'd like to open. Then, one should click "Build" on the top bar of one's computer and press "Build artifacts." Then, they should click on the jar file that shows, either "TankGame.jar" or "RainbowReefGame.jar." After waiting a few seconds, one has completed the build and import of the Tank game or second game in IntelliJ IDEA.

4. How to Run Project

To run the Tank game or second game, complete the build and import steps in the previous section. Now, one should see a folder titled "jar" and open it to reveal the jar in the project. To build the project, right click on the jar and click on the "run" button. Wait a few seconds, and the IntelliJ IDEA should open a new window to view the game on the screen. After that, one has successfully run the Tank game or second game in IntelliJ IDEA.

5. Assumption Made

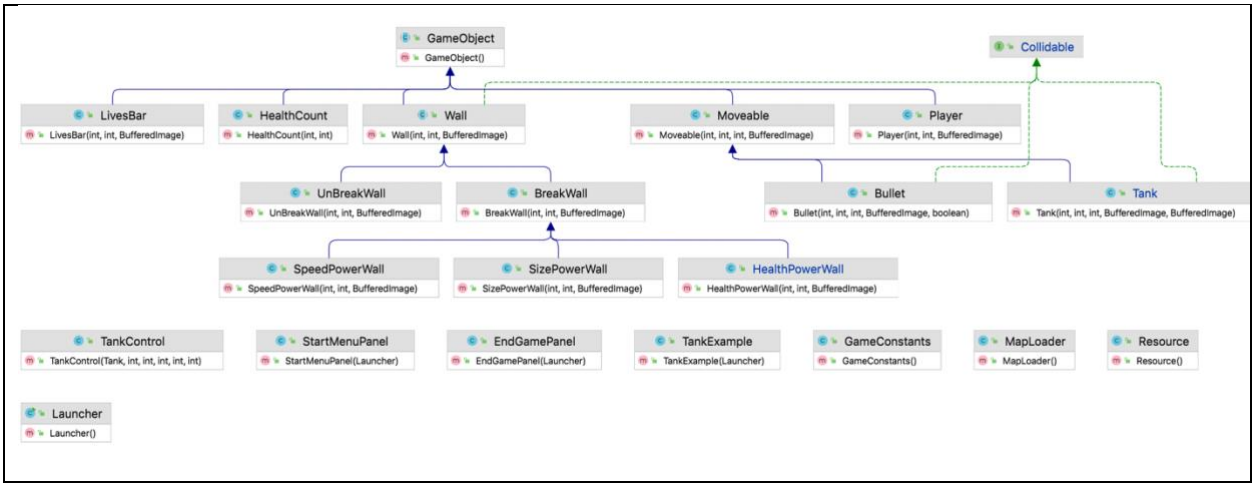
Both: The assumptions I made when designing both projects were that the files are run on a computer with access to keys such as up, down, left and right arrows, and WASD for tank game, and left and right arrow keys for the second game. Additionally, there is an assumption that the display they are running the game on has a height of less than or equal to 960 pixels for tank game, 860 pixels for second game, and a width of less than or equal to 1290 pixels for tank game 1200 pixels for second game.

Tank game specific: I made the assumption that the player will have another person to play with, as it is a two-player game. These two players are working to compete against each other as well, so without a second player the exhilaration of the game dramatically decreases.

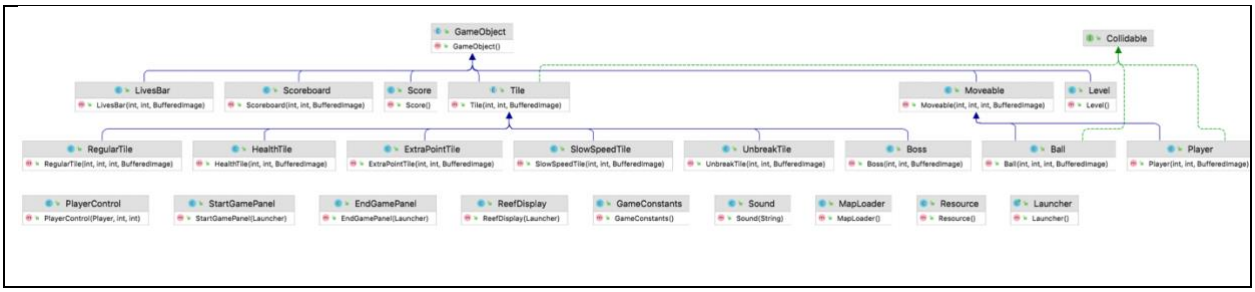
Sundays at Dim Sum (Rainbow Reef Remix) specific: I made the assumption that the player knows English, as I wrote the labels of buttons, scores, and levels in English. While this pertains to the Tank game as well, this is more significant to the Sundays at Dim Sum game which has more labels with English on-screen.

6 Implementation Discussion

6.1 Tank Game Class Diagram



6.2 Sundays at Dim Sum (Rainbow Reef Game) Class Diagram



7 Class Descriptions of Classes Shared Between Two Games

7.1 GameObject Class

Overview: When tackling both games, I first looked at the abstraction of classes at large. All objects that are part of the game will be displayed in one way or another on the screen, so this class that defined that abstraction.

drawImage() in GameObject Class: The GameObject class is an abstract class that has one method, drawImage(). This function takes in one parameter, a Graphics object to draw the object on.

```
public abstract void drawImage(Graphics g);
```

I made GameObject an abstract class and the drawImage() function an abstract function because each child class should have distinct implementations of the function. This is because each object will need to display their own image, whether that be to tell the difference between two special effect blocks or to display a different costume to differentiate two tanks. Apart from just assigning an image, the RegularTile wall on RainbowReef needs to evaluate a set of conditions to see what image it should

display. For instance, if a player has already hit a block twice, it should change the costume of the block every time based on that state.

Moreover, creating this abstraction helps group these functions together when building a display. Rather than having to run the drawImage() function in each object separately, an arrayList with the GameObjects can be created, to which a for loop traverses through the array to run the drawImage() function for each object. This larger picture idea helps set up long term success and maintainability for additional object implementations.

7.2 Moveable Class

Overview: Both the Tank game and Rainbow Reef game required multiple moving objects; the Tanks and bullets in the Tank Game, and platform and ball in Rainbow Reef Game. In a realization of these characteristics, objects that will possess a movable ability are grouped together to extend this Moveable class.

Instance Variables in Moveable Class: To track moveable objects, this class includes instance variables to store the position and direction of the object.

`int x;` `int y;`	The instance variable x tracks the X position of the object in the window. Similarly following, the instance variable y tracks the Y position of the object in the window.

`int vx;` `int vy;`	The instance variables vx and vy track the speed of the object in its respective x or y direction. The speed is useful to include in order to know in which direction they were heading depending on whether the value was positive or negative. Furthermore, these variables are used to assign the speed of an object, to regulate the object to go faster or slower, or stay at a fixed pace.

`int angle;`	In order to know in which direction the movable object is headed, the angle variable is also used to track this. Measured in degrees, this angle variable determines how the object's image should be rotated. In the second game, this is used to turn the ball to face different directions.

`int Rectangle hitBox;` `private boolean hasCollided = false;`	A hitbox is used to track the collisions of the object. This determines whether an object has collided with another object. The Boolean variable hasCollided is used to quickly access whether a collision has occurred for that object.

Important functions in Moveable: While there are getters and setters that will be excluded for redundancy, below are the set of functions that hold significance in defining the class.

```
public boolean checkBorder()
```

The checkBorder() function is used by both games to check whether a moveable object will be out of bounds. This helps to stop a moveable object like a Tank from going beyond the environment boundaries. In this implementation is a series of if-statements that checks whether the x or y value of the object extends the frame it is meant to be in. Depending on whether it is out of bounds or not, it returns a Boolean value: true for out of bounds, and false for in bounds. For direct correction, the x and y values are then set to the maximum or minimum value they can be before they go beyond the boundary, to minimize any lag in the program from making go out of bounds.

```
public void drawImage(Graphics g)
```

The Moveable class extends GameObject, meaning that it will also implement the drawImage() function. This function uses the angle instance variable to ensure it is accurately rotating the image, making the game look more realistic. This function is constantly called throughout the game to update the movement and position of the sprites.

7.3 TankControl/PlayerControl Class

In both games, there needs to be a communication between the keys on the keyboard and our game. In the tank game, this includes the keys pressed to move up, move down, rotate left, rotate right, and shoot. In the Rainbow Reef game, this includes the keys pressed to move the platform left and right. This similarity leads both games to possess a TankControl or PlayerControl class, which implements the interface KeyListener to receive keyboard events. These classes contain the method keyPressed(), which takes in a key event detected. If that key is one of the directional keys or a bullet shooting key, it alerts the object regarding this key that the key was pressed. For instance, pressing the left arrow key in both games would alert the Tank or Platform that left was pressed by calling the toggleLeftPressed() method for them to react to this. Essentially, the TankControl/PlayerControl classes act as a mediator between the user keyboard input and objects in the game to promote smooth communication flow.

7.4 Tile/Wall Abstract Class

In both games, there exists the presence of a stationary object that players can interact with to gain advantages or special effects in the game. In the Tank Game, the Wall class is used to create a more interesting environment for Tanks to navigate in, with breakable, unbreakable, and special power walls. In the Rainbow Reef Game, the Tile class is used for the Ball object to hit in order to increase their score and gain special effects. I made this class abstract because I knew it would take on multiple child classes with different versions of walls and tiles, but at the same time had enough similarity with one another to be grouped under a parent class.

`int x;` `int y;`	Similar to the Moveable class, the instance variable x tracks the x position of the object the instance variable y tracks the Y position of the object. This is helpful in understanding when comparing the position of other objects in relation to the tiles and walls, to generate accurate collisions.

`int state = 2;`	The instance variable state tracks how many more collisions it would need from an object in order to delete itself. Some walls are breakable in the Tank game and Rainbow Reef game, so this would be used to track how many more collisions are left until it breaks. For unbreakable walls, this would still be inherited, yet the state will not be changed.

Additionally, the Tile/Wall abstract class extends GameObject because it is a visible element and component of the game, and implements the Collidable interface because it can collide with others. This class leaves the implementation of the drawImage() function from the GameObject abstract class for implementation in the child classes. This is due to the customization various tiles or walls will need when it comes to portraying themselves, such as being in different colors based on its current state.

7.5 UnbreakTile/UnBreakWall Class

Next, we dive into the child classes that extend either the Tile or Wall abstract classes. UnbreakTile and UnBreakWall represent the blocks that cannot disappear and are permanently residing in their environment, no matter how many collisions they interfere. As a child of the Wall class, it implements one method, the drawImage() function that the Wall class did not implement. It also contains a constructor that uses the super keyword to run the constructor of the parent class.

Although this is a short class, it is crucial for the development of the game. The unbreakable ability of the wall provides structure within the game and sets boundaries.

7.6 HealthTile/HealthPowerWall Class

Both games had their own special effects, there are some overlaps between these games to create a similar special effect in both games. In the HealthTile/HealthPowerWall class, the mission is that upon colliding a bullet or ball to this, it will grant the player a special power in their health. For the Tank game, the HealthPowerWall will reset a player's health status to three full bars of life each with 200 health points. For the Rainbow Reef Game, the HealthTile will give one extra life to a player, but maxes out at three lives. However, these special effects will be handled in the Ball or Bullet class. This is because this class does not have and should not have control over classes other than itself, as it lines towards the Single Responsibility Rule of only being a wall, not interfering with a Player or Ball class.

7.7 SpeedPowerWall/SlowSpeedTile Class

This is another special effect block that has similar motives with one another, both regarding speed. These blocks extend the Wall or Tile class, yet also remain fairly simple in its implementation. Like the other child classes of Wall or Tile, they take a minimal approach in code and leave the function's effect up to the Ball or Bullet class to handle. The Bullet object will detect the SpeedPowerWall and set its speed to 2.5x greater than its current speed, and the Ball object will detect the SlowSpeedTile and set its speed to a slower pace.

7.8 Collidable Interface

After determining the Movable abstract class and Stationary objects (Wall/Tile), I decided to view the objects from a different light as well: collidability. Some objects needed to detect collisions with other objects, like a platform to ball and bullet to tank. Thus, this collidable interface is used to group these classes together. I chose to create an interface rather than an abstract class because the trait a common functionality that can be found in unrelated classes like Walls and Bullets. For this reason, I created three abstract functions in this class for objects who collide to implement.

```
public abstract Rectangle getRectangle();
```

A key component to determining collisions is understanding if two objects are touching or not. The getRectangle() function will test this idea by returning the hitBox of collidable objects, which can be used to see if they are overlapping on not.

```
public abstract int getX();
public abstract int getY();
```

The next two functions are the getters for X and Y. Although these may seem like a nuisance to include, these getters are extremely necessary in detecting where the position of a collidable object is. By including in the collidable interface as well, it helps remind and ensure that classes who are collidable to implement these functions.

7.9 GameConstants Class

In both games, there exists variables that never change but are relatively abundant in the program. These variables also must be universally accessed by classes in the program, so we declare these as public and static. The GameConstants class is used to put these public, static, and final variables all in one place.

```
public static final int WORLD_WIDTH = 2010;
public static final int WORLD_HEIGHT = 2035;
```

Some of these constants include dimensions for the game height and width as shown above. This class organizes these constants for easy reference and maintainability. Although one may think that numbers that never change can be more easily just directly inserted into code, creating a final variable for this value will be especially useful if this final variable ever needs to be adjusted. Instead of having to go through each time we directly inserted a number, we can just change the variable value in the GameConstants class to update all the times that value has been used.

7.10 MapLoader Class

The Tank game and Rainbow Reef game both display their environments from the information they receive by loading in a file. For the Tank game, characters are read in a grid-like format to produce Wall objects in certain positions of the environment. In the Rainbow Reef game, a similar approach is taken but for Tile objects instead.

```
public void loadMap(ArrayList<Wall> walls)
```

Loading a file can be lengthy, so placing it in a MapLoader class helps split up the responsibilities of certain classes and make the code more readable. The MapLoader class contains one function, loadMap(), which takes in an ArrayList of walls that it will add to based on the information in the file. It uses an InputStreamReader and BufferedReader to read the values from a file.

```
case "2":
    walls.add(new BreakWall(curCol*30, curRow*30,
Resource.getResourceImage("breakWall")));
    break;
```

Then, it uses a switch function to read each character and assign add different wall or tile child classes to the ArrayList. As shown above, a “2” in a file will add a new BreakWall object to the list. This switch function makes it easy to continue adding more ways to read a file, as switch cases can easily be added and removed.

7.11 Resource Class

Another lengthy yet crucial part to building the environment is loading the files of a game. The Resource class is used to organize this process by holding the files of a game all in one place.

```
private static Map<String, BufferedImage> resources;
```

This class stores a Map of String to BufferedImage, which consists of a label key that points to the loading of an image. This Map is private in order to encapsulate the files it holds, so that no other class can alter or change its contents. It is also static for other classes to conveniently access without the need of creating an instance of Resource. When developing a project, this Resource class is essential to have

in order to know where to go when a file is missing or needs to be added, as opposed to being dispersed throughout the program.

7.12 EndGamePanel Class

Both games have a way of win and losing, so both have an EndGamePanel class to displays the end of a game on a user's screen. This class is a child class of JPanel, which is used in order to organize various layouts that show on an ending screen.

```
private JButton start;
private JButton exit;
```

The EndGamePanel contains two JButtons a restart button and an exit button. These are displayed to help bring a call to action for the user to either replay the game or quit. It also contains a BufferedImage that displays the background of the class.

```
public EndGamePanel(Launcher lf)
```

In its constructor, it initializes the JButtons, JLabels, and menu background and adds these to its JPanel. This means that when the EndGamePanel is initialized, it will have already had its display created in this class. It takes in the parameter of a Launcher class, in order to call its functions like closing the game or setting up a JFrame.

7.13 StartMenuPanel Class

Similar to the EndGamePanel class, the StartManuPanel class is a child class of JPanel to display a view for the very beginning of the program. It contains buttons start and exit as well, and its constructor takes in a Launcher to set its buttons to call the start or exit of a game.

```
private JButton start;
private JButton exit;
```

```
public StartMenuPanel(Launcher lf)
```

The similarity between the StartMenuPanel class and EndGamePanel class make it easier for another class to organize and call these functions, as their syntax will primarily be the same. For the developer wishing to make changes to either class, they will be more at ease because these classes are cohesive and follow a similar style convention.

7.14 LivesBar Class

The Tank game and Rainbow Reef game have an end goal but also a way to end a player's game early. Each player in both games has three lives, and once those lives run out, the game automatically ends. The LivesBar class is used to represent player lives, and extends GameObject as it is a component of the game to be displayed.

```
private int x, y;
private int playerLives = 3; // Each player is given 3 lives
BufferedImage lifeImage;
```

The LivesBar class has instance variables x and y, in order to know where it should be positioned on the screen. It contains a instance variable representing the playerLives, which it will use to be increased and decreased based on the occurrences in the run-time of the game. It also contains a lifeImage, used to be displayed as the heart in the game.

```
public void drawImage(Graphics g)
```

It also has a drawImage() function that uses a for loop to print out a heart image for every life there is, each time moving the x position of every heart so they are next to each other rather than overlapping each other.

7.15 Scoreboard/Player Class

With the many tracking objects in each game, the Player class in Tank game and Scoreboard class in Rainbow Reef game possesses a has-a relationship for these objects in order to handle communication between these trackers and keep them grouped in one class.

```
private LivesBar livesBar;
private HealthCount healthCount;
```

For the Player class in Tank Game, this class has a LivesBar object and HealthCount object. Since LivesBar and HealthCount are interrelated, the Player class steps in to guide the process of connecting communication between the two. For instance, when the HealthCount becomes 0 points, the Player class subtracts a life from the LivesBar and resets the HealthCount object back to 200 points.

```
//Tank Game

public void resetHealth(){
    livesBar.setLives(3);
    healthCount.reset();
    isDead = false;
}
```

```
//Rainbow Reef Game

public void reset() {
    livesBar.setLives(3);
    isDead = false;
    level.setLevel(1);
}
```

Both also contain a reset() or resetHealth() function, used to reset the tracker objects it contains. This is helpful because the Scoreboard and Player class manage all of the tracker objects at once, so there is no need to individually make more calls for each object.

8 Class Descriptions of Classes Specific to Tank Game

8.1 Bullet Class

Getting specific to Tank game, the Bullet class represents the bullets Tank objects shoot in the game to destroy blocks and the opponent Tank object.

```
public class Bullet extends Moveable implements Collidable
```

The Bullet class extends Moveable because it moves around the environment and implements Collidable because it can collide with other objects, namely the Wall and Tank objects in the Tank game.

```
public void moveForwards()
```

A bullet's form of movement is a linear path, and it disappears once it collides with an object. The moveForwards() function is used to calculate and put into effect this pattern of motion. It looks at the current directional the player is pointing to, then continues to move in that direction until the object receives information that it has collided. This function uses the checkBorder function and resets the hitBox location every time, to make sure that the Bullet will disappear once it hits a border, which isn't an object, as well.

```
boolean isMassiveBullet;
```

To aid one of the Tank's special effects, shooting a massive bullet, it has a Boolean variable isMassiveBullet to know whether this object should be massive or not.

8.2 Tank Class

In Tank game, a Tank is able to move around its environment and shoot bullets. In this multi-player game, it is also the way players control their health.

```
private ArrayList<Bullet> ammo;
```

Since the Tank object can shoot bullets, it contains an ArrayList of Bullet objects titled ammo, for ammunition. Tank objects own Bullet objects so that it is clear whose bullets are whose. This will reduce the amount of comparisons Tanks will make when checking whether a Bullet object interferes with it, as it will only need to check with the other player's ammo ArrayList.

```

```

```
public Player player;
```

The Tank also owns a Player object which is used to keep the score of that player. With two tanks in the game, there will exist two Player objects, each tracking a different user and tank. The ownership of the Player object helps attribute the Tank to its score easily, and allows the Tank to behave certain ways depending on the player by looking into its stored data.

```
public void update();
```

Each Tank needs to update its behaviors depending on which key on the keyboard is pressed at all times. The update() function helps with this by checking whether the up/down/left/right/enter or w/a/s/d/space keys are pressed to run certain commands. Additionally the update() function loops through a player's ammo ArrayList to delete any Bullet objects from its list that has collided. This helps shorten the size of the ArrayList and keep track of only the relevant Bullet objects.

```
public void collidedWithAmmo(Collidable obj)
```

This function is used to appropriately make comparison with a Collidable object in the program. It uses a for loop to iterate through its list of Bullet objects and detect any collisions it makes with objects. It then uses if statements to compare whether objects of a specific type have collided, and if so, to make specific instructions to adjust the program.

```
public static void willCollide(Collidable obj1, Collidable obj2)
```

The Tank object also contains a function to compare the collisions of two Collidable objects, obj1 and obj2. This function uses the hitboxes and intersect() function to see if two objects are touching each other. If they are, it runs a series of if-statements to see what type they are, and performs results accordingly. This willCollide function is abstracted well, as any two Collidable objects can be compared using just one function implementation. Because of this, it makes it easier to maintain and scale the code as new features develop.

8.3 SizePowerWall Class

One special effect that was distinct to the Tank class was an effect to increase the size and damage of the player's next three bullets. The wall to gain this special effect was SizePowerWall. This class extends the Wall abstract class to inherit variables like x,y,state, hitBox, and drawImage(). Because of this, there is little needed to implement in this class itself. The Tank class will be the class that puts these effects into action, so the SizePowerWall class is only needed to differentiate itself when others realize they collided into it.

8.4 HealthCount Class

One tracker that is only present in the Tank game is the tracking of one's health status in a bar form. To do this, a HealthCount class is created to represent the amount of health left before a player loses one of their lives.

```
private int x, y;
private int healthCount = 200;
```

This class has variables x and y to help position the healthCount on the screen, and a healthCount instance variable set to 200 points at the very start, which will track the player's health.

```
@Override
public void drawImage(Graphics g) {
    …
    g2d.fillRect(x, y+10, healthCount, 30);
    g.setColor(new Color(56, 173, 105));
    …
    g.drawString(String.valueOf(healthCount), x+healthCount + 10,y+30);
}
```

The HealthCount class extends GameObject, as it will be displayed on screen as a measure of the player's health. In its inherited draw image function which it must implement, it creates a frame with a gray background and on top, a green rectangle that shrinks and expands its width depending on how many health points a player has. This visual is helpful in displaying a form of a bar chart to show users the numerical values of their score during game time.

8.5 TankExample Class

The TankExample class acts as a driver class for the Tank game, connecting the GameObjects in one place. It extends JPanel and implements Runnable, to create a visual display of the moveable and stationary objects of Tanks, their bullets, and walls.

```
private Tank t1;
private Tank t2;
public static ArrayList<Wall> walls;
```

This class contains two Tanks, one to represent each player. These tanks are responsible for their own health tracker and bullets. This class also contains an ArrayList of Wall objects, to be used to keep track of the walls left in the environment.

```
public void run(){
    try {
        …
        while (true) {
            …
```

```
            if(gameOver()){
                …
                return;
            }
        }
    } catch (InterruptedException ignored) {
        …
    }
}
```

The run() function is what ties the game together, by running a while loop of updating Tank and bullet movement, searching for game time collision, redrawing the display, pausing for a blink of time, and checking if the game should end. This function is deliberately made to hold only under twenty lines of code, which has been done through abstraction and dividing the responsibility of tasks to other classes.

```
public void searchForGametimeCollision()
```

This function takes is responsible for detecting any game time collisions, now that the objects are all present. It uses for loops and if statements to run collision checks on walls, tanks, and bullets. This is the sole and overarching method used for all gametime collision, and has been able to do so through the use of the Collision interface on objects, so that they can easily be compared with one another from their getRectangle() function.

```
public void paintComponent(Graphics g)
```

Another notable function from the TankExample class is its paintComponent method from extending the JPanel class. Inside it, it sets up the drawing of all GameObjects. This function groups together the GameObjects to run the drawImage() function through a for loop. It also creates three BufferedImage displays, one representing the left half of the screen to follow the first tank, the right half of the screen to follow the second tank, and a minimap in the center of the screen. The minimap is created by decreasing the scale of the display by 90% and positioning it in the middle of the screen.

```
public int getRevisedX(int x)
public int getRevisedY(int y)
```

These getRevisedX and getRevisedY methods get the revised X or Y coordinate in context of the world to game screen ratio. Depending on the x and y position of the tank, the method will return and x and y position for the left half and right half of the screen to focus on, in order to keep both the Tank objects in sight.

9 Class Descriptions of Classes Specific to Second Game

9.1 Ball Class

The ball class is similar in to the Bullet class in the tank game, and extends Moveable because of its ablility to move around, and implements Collidable as it can collide with other objects.

```
boolean touchedGround = false;
int bossCollisions = 0;
```

Some notable instance variables in this class include the touchedGround Boolean variable, which represents whether the ball has touched the ground or not. This is crucial in communicating whether we have lost a life. The boss collisions instance variable mentions whether we were able to collide with a certain number of bosses, to know when is the right time to move onto the next stage.

```
public void move()
```

The move function of the ball class checks whether the ball has collided with anything. Since it moves in a linear direction until it collides, this function simply increases the x and y in their respective amounts in relation to the direction incrementally. Then, it updates the hitBox to make sure that the other objects who reference the hitBox of the ball have accurate information regarding its location.

```
public void bounceBack(Collidable obj)
```

The bounceBack function makes the ball bounce back from an object depending on the type of object it collides with. The parameter is a Collidable obj, which can be a player or type of tile. If the ball bounces to the left side of the player, it is going to head in a general left direction. If the ball bounces to the right side of the player, it is going to head in a general right direction. However, these directions are all general and random, so one cannot specifically pinpoint the angle to which the ball will head.

9.2 Player Class

The player class is used to control the platform, or the food dish of the game. In this game, it uses left and right arrow keys to control itself. It cannot go across the boundaries, however. There is only one collision a player must deal with: the ball. If it collides with the ball, it will grant the ball an increase in speed. It also uses its update() function to check whether a user has pressed a left or right key. Moreover, when a ball collides with a player, the player will play a pop sound from the Sound class.

9.3 ReefDisplay Class

The ReefDisplay class is quite similar to the TankExample class in Tank Game, where it acts as a driver class for the Rainbow Reef Game. This class involves the player, ball, scoreboard and other objects in the game.

```
public void loadMap(String mapName)
```

Although there are many methods in this function, here are a few unique ones that have not been touched upon from overlap in the Tank game. This function, loadMap(), specifically loads the map of a mapName entered in the parameters of the function. This is essential for games like Rainbow Reef with multiple levels, in order to seamlessly swtich from one level to the next. This function creates a new mapLoader object, sets the mapName to load the map in with the argument passed in, then loads the map with the tiles ArrayList that is present in the class.

```
public void updatePlayerInfo()
```

Another function unique to the Rainbow Reef Game is the updatePlayerInfo() function, which helps draw the right bar of the screen shown in the game. It uses JLabel objects to write the score, ball speed, current level, and provides a key to the special effect blocks in that class.

```
public boolean checkForBallCollision(Ball ball, Collidable obj)
```

There is also a checkForBallCollision function, which compares the collisions between a ball and a collidable object. The ball is the only object in this game that interacts or collides with more than one object, so that is why there is a type dedicated to Ball in the parameter as opposed to two Collidable types in the Tank game method. By doing this, it makes it easier to use because there is no need to test whether a Collidable obj is a ball, and then to cast the Collidable object to a ball. This function compares the ball's hitBox with the hitboxes of other objects. If they intersect, then the bounceBack() function of the ball is called. This method helps to check all collisions during runtime in a single method.

9.4 Sound Class

In the Rainbow Reef Game, I decided to add sound for a special addition and more dimensional touch to the game. The Sound class uses AudioInputStream and Clip to take in an audio and play it.

```
public Sound(String fileName)
```

The constructor of the Sound class requires the filename, which is how the object will access a sound file. The sound files in this program are stored in the resources folder and in an WAV format. The constructor uses a try-catch to attempt to open the file and convert it into an AudioInputStream to store as a Clip object. It may run into several exceptions, including UnsupportedAudioFileException, LineUnavailableException and IOException.

```
public void play(){
    clip.setFramePosition(0);
    clip.start();
}
```

In order to play the sound, a simple play() function is used. It sets the clip's frame position to zero, to make sure the clip starts from the beginning each time it is played. Then, it starts the clip.

10 Project Reflection

Looking back on this project, I have learned an abundance on how to create a structure for a multiplayer and single player game. I have also learned how to effectively communicate between different classes, and how to interpret and execute specific commands from a file, like a map used in both games. I also take note of how it can seem unnecessary to abstract at first, but it is so important to do because it makes adding extra additions into the game much easier. When creating my Tank game, I abstracted a lot and that has helped me in creating the second game. I also learned the importance of packages to sort and organize my files. Prior to this, I would not use packages and wasn't too clear about why they were there. Additionally, I was able to have a sense of creativity to explore my with my project.

11 Project Conclusion/Results

From doing these projects, I feel more confidence in my ability to code a larger program and set up the structure of something that has more classes. Creating the Tank game and second game showed me that although it can seem intimidating to create a game at first, it is a lot easier than one may think once we break down the problem into smaller pieces. I also feel more familiar with JFrame and JPanel, and will definitely look towards using those to create more games in the future.