Developers' Guide

Term Project

CS413 Spring 2016 Professor Ilmi Yoon

March 23, 2016

Jeffrey Ilar

Table of Contents

Introduction:

The term project of this semester was to create two games. The first is a two player game where the objective is to destroy the other player's tank. The second game was one of our own choosing. For my project I decided to program the brick breaker game Rainbow Reef. The main focus of these term projects was to test of knowledge of inheritance and reusability.

Given Resources:

We were given two complete game codes for an airplane shooter game. We were also provided documentation from a previous student's term project. Throughout the remaining weeks Professor Yoon gave us various implementation hints regarding both game projects during lecture. Images and sound files were also provided for both games.

Scope of work:

For this project we were allowed to reuse the game engine of either of the Plane Games From there were we to extend the game engine classes in order to program both games.

Assumptions:

Development of program files were performed in Netbeans IDE 8.0.2 environment on Windows. My term project uses the game engine of Lowell's Plane Game as a base.

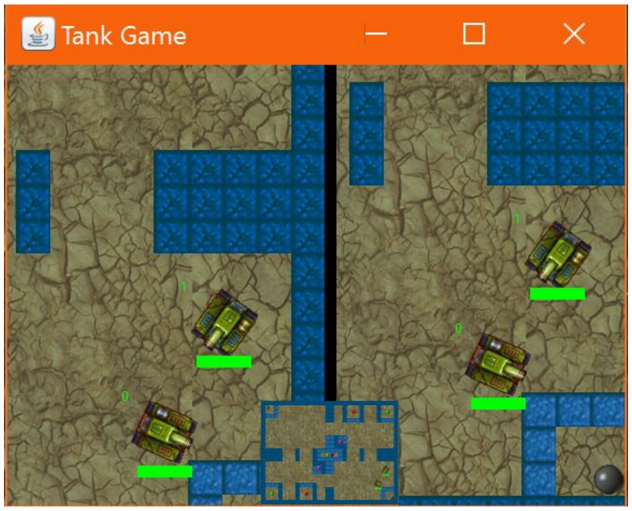
How to Use Jar File:

Simply open TankGame.jar or RainbowReef.jar to begin playing either game.
Some things to note:

-TankGame.jar must be opened in the same directory as Battleground.txt

-RainbowReef.jar must be opened in the same directory as Reef1.txt and Reef2.txt

Tank Game:



The green bar at the bottom of both tanks indicates the tanks' life. When it is fully gone the tank explodes. The number next to the tank indicates how many times you have defeated the other player.

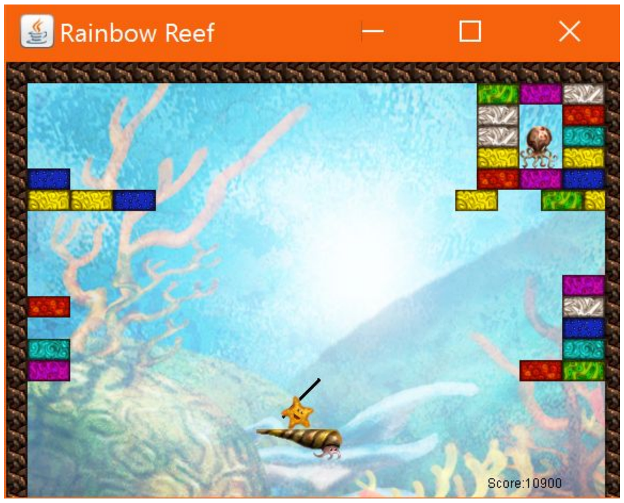
Player 1 is controlled with the arrow keys. To shoot bullets press the shift key. Player 2 is controlled by wasd keys, and can shoot bullets with space.

There are multiple power ups in my game in the form of colored balls. Their effect are as follows:

Grey	-Incraesed Damage
Green	-Restore Health
Red	-Place Mines
Purple	-Ghost Bullets (bullets go through walls)
Blue	-Ghost Tank (ability to pass through walls)

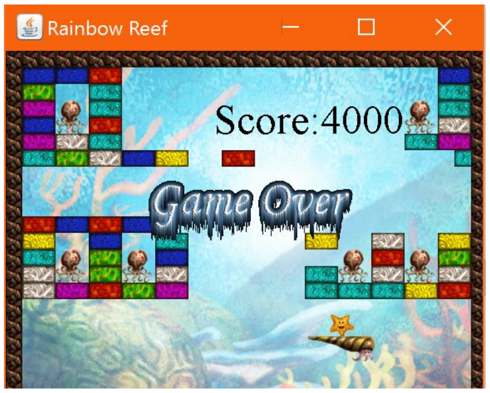
Lastly, light blue walls are destructible and can be removed with enough bullets. They do respawn however.

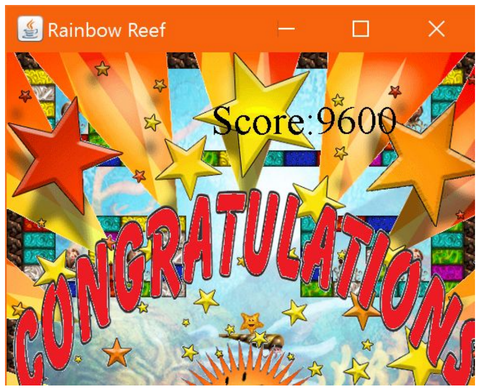
<u>Rainbow Reef:</u>



The player controls the bar at the bottom (named Katch) using the arrow keys. Left and Right keys move Katch left and right respectively. Using up or down changes the starfish's (Pop) trajectory.' The black line indicates Pop's current trajectory before the launch. Lastly can be launched by the player with the press of the spacebar. Spacebar doe not do anything if pop i

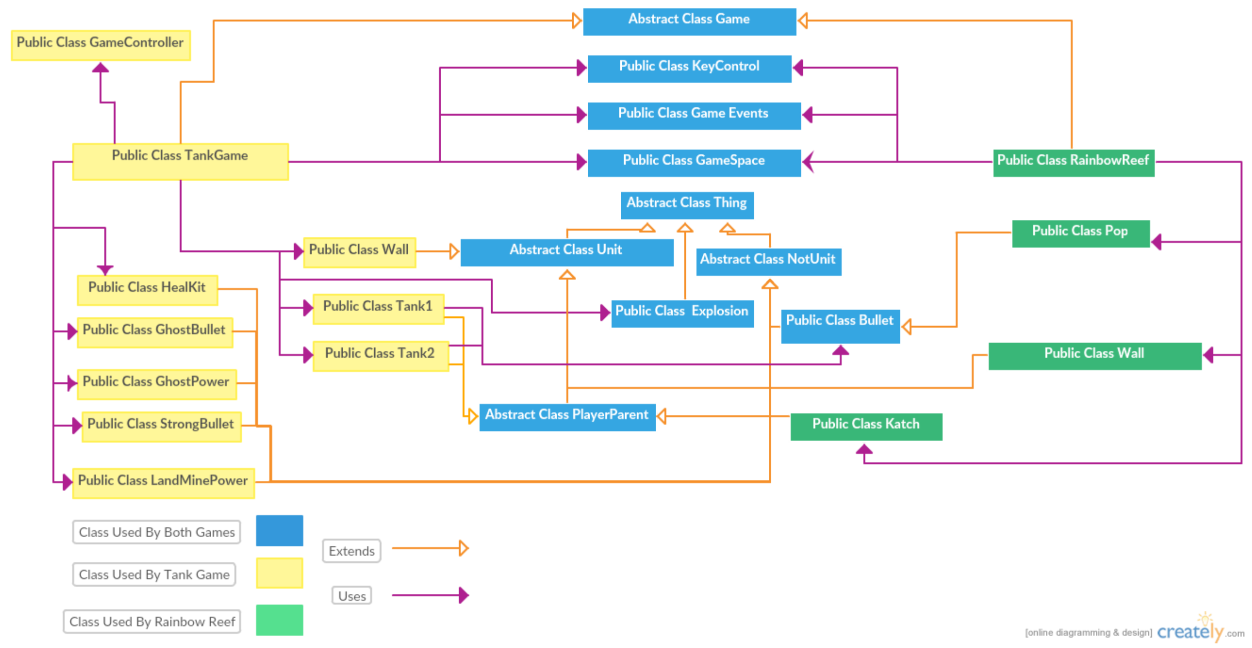
Remaining player lives (not shown in above screenshot) appear on the bottom left portion of the screen. Current player score is on the bottom left. Score increases with each enemy and block hit by Pop.





Appropriate screens appear when the player loses all lives, or beats the final level.

Class Diagram:



Shared Class Details:

In this section I will go over the function of the classes shared by both games. These shared classes act as the game engine. They come directly from the code provided from Lowell.

Abstract Class Game:
Game extends the JApplet class. It is responsible for initialized the starting parameters of the game, starting the game, and lastly running the game.

Public Class KeyContol:
KeyControl acts as a keystroke reader. It helps determine when a key is pressed and when it is released.

Public Class GameEvents:
GameEvents extends the observable class. It is watched by all Thing objects, so they can update appropriately.

Public Class GameSpace:

The purpose of Gamespace to update the java application by drawing both the background as well as any existing thing objects. Two Important methods in this class are drawBackground() which creates the game's background, and drawHere() which loops through an Arraylist of Arraylist Things, to draw all the things on the gamespace. This class is used in the drawAll() of class TankGame and class RainbowReef.

Abstract Class Thing:
The Thing class represents all objects to be drawn on the java application. All things are capable of being collided with, but not all things are able to cause collisions.

Abstract Class Unit:
The Unit Class extends Thing. All Units are capable of causing collisions, events that happen when a thing touches the Unit.

Abstract Class PlayerParent:
PlayerParent extends the unit class. The unit classes contain added methods that allow for unit movement due to keystrokes, as well as the ability for the unit to "fire" or cause an action.

Abstract Class NotUnit:
NotUnit extends the thing class. The NotUnit class incapable of detecting collisions, however Units can still collide with NotUnits and detect a collision themselves.

Public Class Bullet:
The bullet class extends the NotUnit class. The bullets class serves as moving projectiles that Unit's can collide into. When a Unit collides with a Bullet instance enough times it will be marked as done and dissapear.

Tank Game Class Details:

This section details classes specific to the Tank Game.

Public Class TankGame:
The TankGame class extends the Game Class. The Tank Game class creates all instances of Units in the game. On startup it loads all needed resources, initializes all variables, and sets up the map which included walls and power up placement. TankGame's method drawAll() draws everything on screen, which included both the healthbar as well as player score.

Public Class GameController:

The only purpose of the class GameController is to create both tank instances. For some reason I found that the players were not able to move unless I made thier instance within this class.

Public Class Wall:
Wall Extends the Unit Class. The Wall Class is a non movable unit. Once the wall unit takes enough damage it will explode. After a period of time it will respawn. To create indestructible walls, I changed the max damage to an incredible high number, 9999.

Public Class Tank1 and Tank2:
Both of these classes extend the PlayerParent. Instances of these classes are what the player controls. It is able to collide with walls, power ups, the other player's bullets, and the other player themselves. Both of these Tank class's work exactly the same, except for the bullet instance they create. A mistake I made was not making use of reusability to only have one Tank Class.

Public Class HealKit:
Healkit extends the NotUnit class. When a tank collides with a HealKit, its damageto variable is reset back to zero. It acts as a power up to reset player health.

Public Class GhostBullet:
GhostBullet extends the NotUnit class. When a tank collides with a GhostBullet power up the bullets it shoots go through walls while the power up is active. I achieved this by disabling the collision check between all wall instances and the Tank's bullet.

Public Class GhostPower:
Ghost Power extends the NotUnit class. When a tank collides with this power up it is able to pass through walls while the power up is active. To do this, I disabled the collisions between the tank and walls in the tank's move() method.
Public Class StrongBullet
Strong Bullet extends the NotUnit Class. The StrongBullet power up changes the type of bullet instance created by the tank. The new bullets have increased damage compared to the regular bullets.

Public Class LandMinePower
LandMinePower extends the NotUnit Class. The LandMinePower class is a power up that changes the type of Bullet instance created by the tank. After getting the power up the tank will place non-moving bullets (up to 5 before reverting back to normal bullets) that stay on screen until the other player tank collides with them. To achieve this I set the speed of the new bullet instance to 0.

Rainbow Reef Class Details:

This section details classes specific to the Rainbow Reef Game.

Public Class RainbowReef:
RainbowReef extends the Game Class. This class is responsible for initializing, and updating the Rainbow Reef Game. It all instances of Things in the game, which includes the level design. It also keeps track of the player score, the remaining amount of player lives, and changes the screen based on if the player completed the level or not.

Public Class Katch:
Katch class extends the player parent. The Katch class is what the player controls in game. Unlike the Tank classes it is only able to move left or right. Pressing up or down changes the trajectory of Pop before it is shot.

Public Class Pop:
The Pop class extends the Bullet Class. Pop acts like a projectile, except it bounces on collision rather than disappearing. Unlike the Bullets of the Tank Game, Pop's Y position moves in relation to gravity.

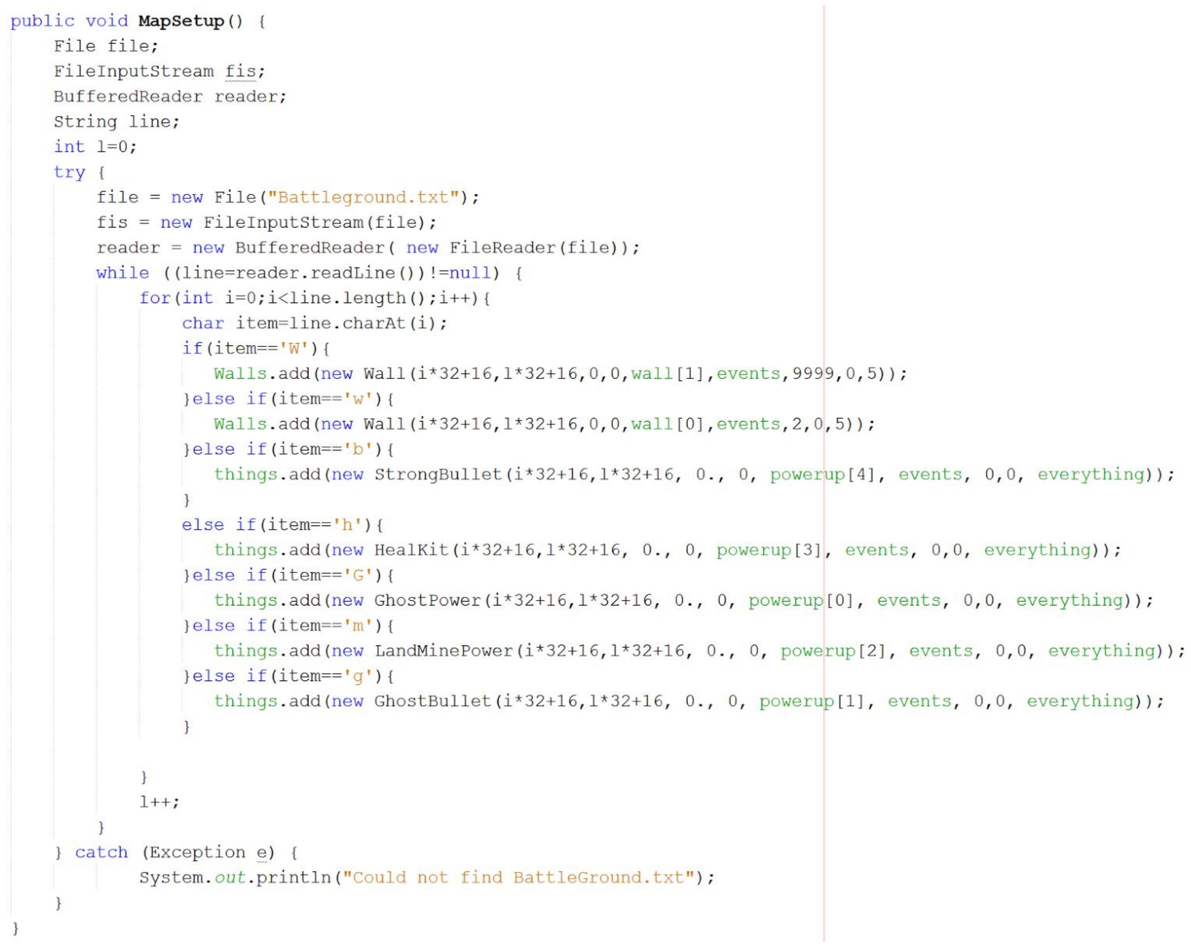
Public Class Wall:
Wall extends the Unit class. In rainbow reef instances of the wall class represent everything in the game that is of type Thing, and the player has no control either. The boundary walls, the octopus enemies, and the bricks are all instances of walls with different score values and images.

Struggles:

This section highlights specific problems I came across while developing both games.

General Struggles:

Generating a Level by Reading a File:

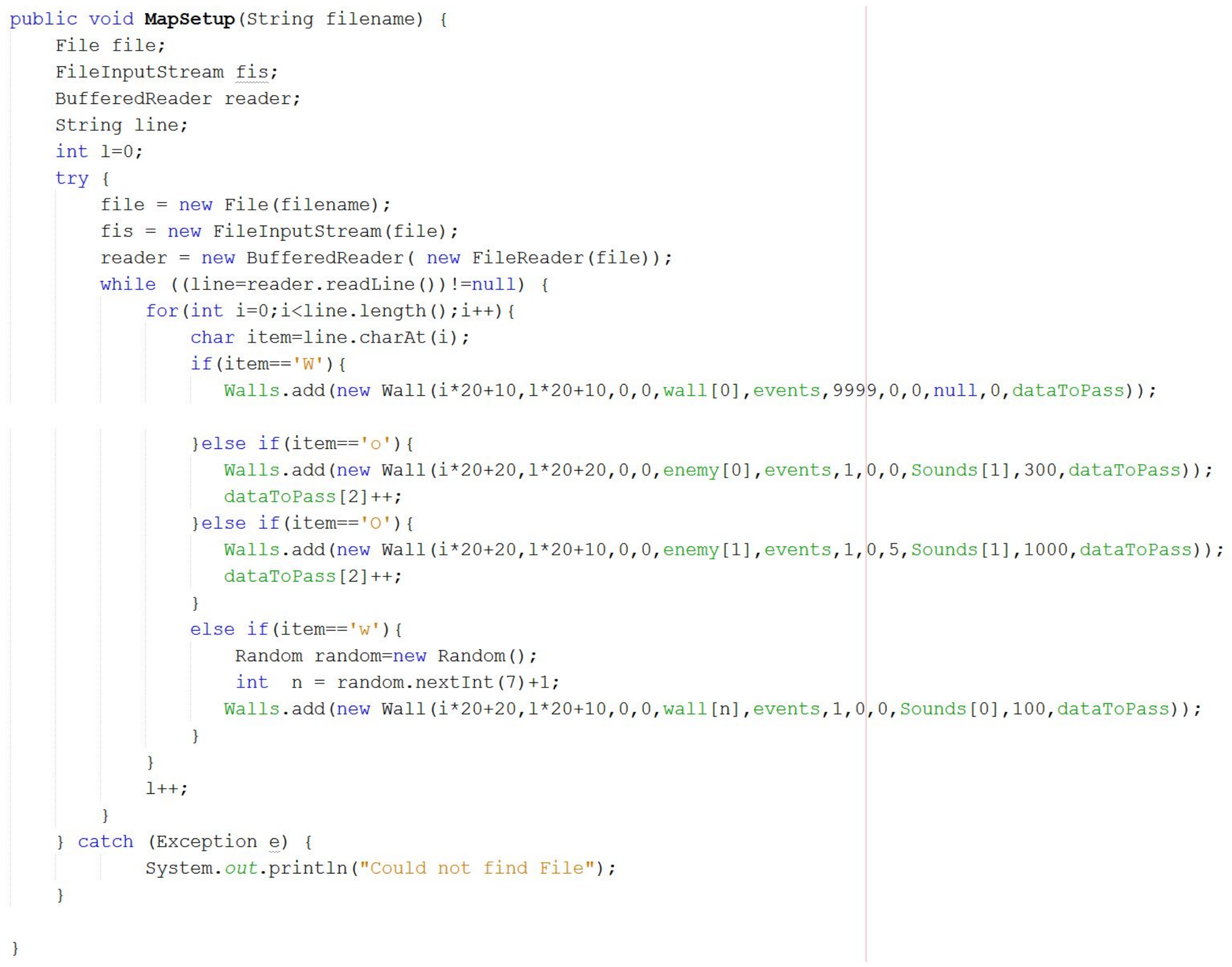


```java
public void MapSetup() {
    File file;
    FileInputStream fis;
    BufferedReader reader;
    String line;
    int l=0;
    try {
        file = new File("Battleground.txt");
        fis = new FileInputStream(file);
        reader = new BufferedReader( new FileReader(file));
        while ((line=reader.readLine())!=null) {
            for(int i=0;i<line.length();i++){
                char item=line.charAt(i);
                if(item=='W'){
                  Walls.add(new Wall(i*32+16,l*32+16,0,0,wall[1],events,9999,0,5));
                }else if(item=='w'){
                  Walls.add(new Wall(i*32+16,l*32+16,0,0,wall[0],events,2,0,5));
                }else if(item=='b'){
                  things.add(new StrongBullet(i*32+16,l*32+16, 0., 0, powerup[4], events, 0,0, everything));
                }
                else if(item=='h'){
                  things.add(new HealKit(i*32+16,l*32+16, 0., 0, powerup[3], events, 0,0, everything));
                }else if(item=='G'){
                  things.add(new GhostPower(i*32+16,l*32+16, 0., 0, powerup[0], events, 0,0, everything));
                }else if(item=='m'){
                  things.add(new LandMinePower(i*32+16,l*32+16, 0., 0, powerup[2], events, 0,0, everything));
                }else if(item=='g'){
                  things.add(new GhostBullet(i*32+16,l*32+16, 0., 0, powerup[1], events, 0,0, everything));
                }

            }
            l++;
        }
    } catch (Exception e) {
            System.out.println("Could not find BattleGround.txt");
    }
}
```

MapSetup Method of TankGame

To generate levels through a file I added a method called MapSetup(). MapSetup looks for a specific file and then uses a BufferedReader in order to read it line by line. A for loop is then used to go through the line character by character. Depending on the character read, an instance of a Thing is created and placed at a certain position based on where it was was read in the file.

The MapSetup method of Rainbow Reef class was modified to take in a String argument. The method then searches for a file with the name of the string and uses it to create the level. By doing this I was allowed to call MapSetup() multiple times to set up different levels in the same game.



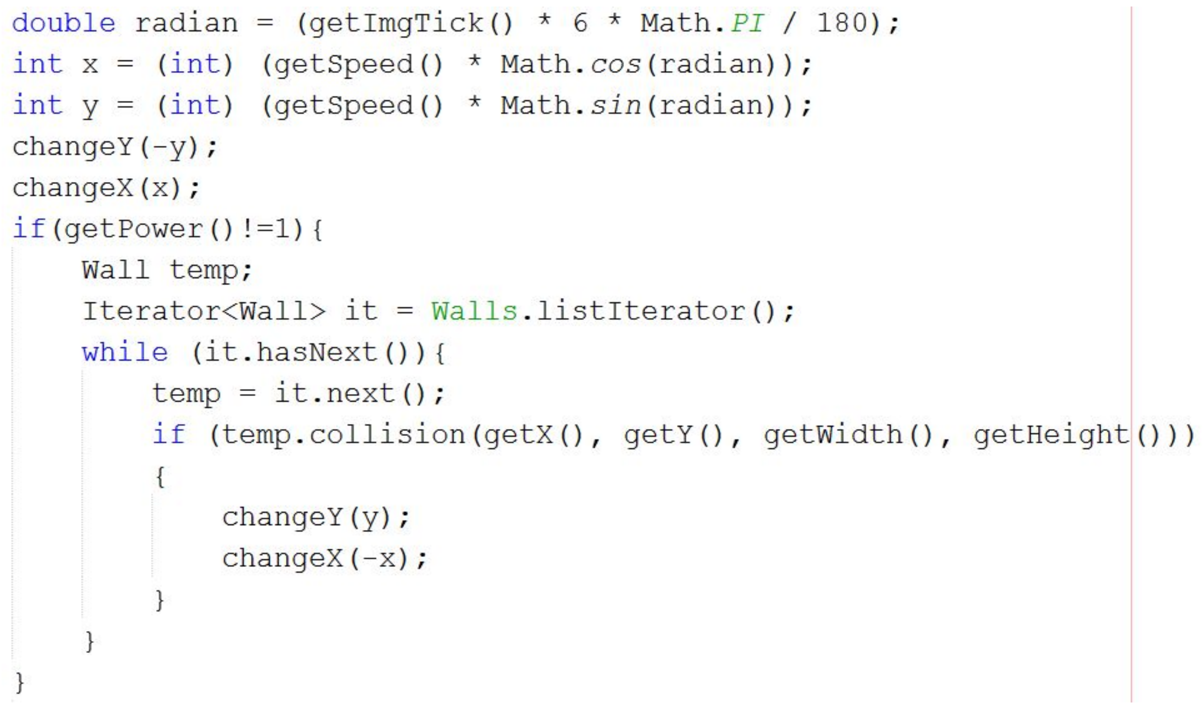
```
public void MapSetup(String filename) {
    File file;
    FileInputStream fis;
    BufferedReader reader;
    String line;
    int l=0;
    try {
        file = new File(filename);
        fis = new FileInputStream(file);
        reader = new BufferedReader( new FileReader(file));
        while ((line=reader.readLine())!=null) {
            for(int i=0;i<line.length();i++){
                char item=line.charAt(i);
                if(item=='W'){
                   Walls.add(new Wall(i*20+10,l*20+10,0,0,wall[0],events,9999,0,0,null,0,dataToPass));

                }else if(item=='o'){
                   Walls.add(new Wall(i*20+20,l*20+20,0,0,enemy[0],events,1,0,0,Sounds[1],300,dataToPass));
                   dataToPass[2]++;
                }else if(item=='O'){
                   Walls.add(new Wall(i*20+20,l*20+10,0,0,enemy[1],events,1,0,5,Sounds[1],1000,dataToPass));
                   dataToPass[2]++;
                }
                else if(item=='w'){
                    Random random=new Random();
                    int  n = random.nextInt(7)+1;
                   Walls.add(new Wall(i*20+20,l*20+10,0,0,wall[n],events,1,0,0,Sounds[0],100,dataToPass));
                }
            }
            l++;
        }
    } catch (Exception e) {
            System.out.println("Could not find File");
    }

}
```

MapSetup Method of RainbowReef

Tank Game:

Tank Gets Stuck in Wall:

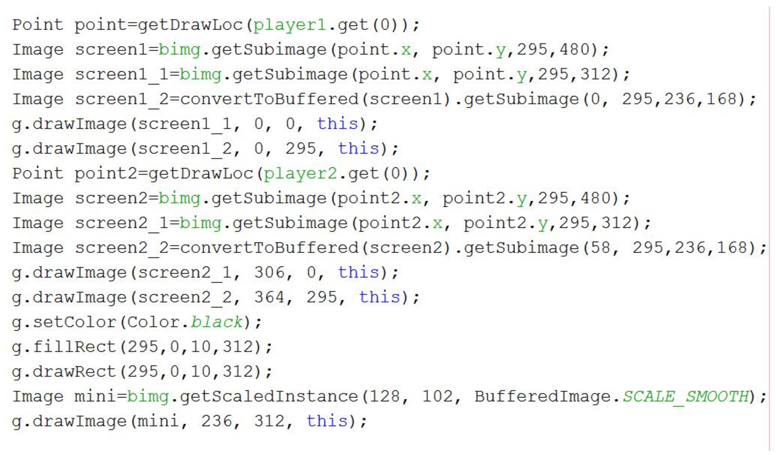
One problem I faced with the Ghost Tank power up was the fact if the player is in a wall when the power up ends he becomes stuck and unable to move unless the wall gets destroyed. To remedy this, I added an if statement. This statement forces the tank to move backwards if it detects a collision between the the tank and a wall.



```java
double radian = (getImgTick() * 6 * Math.PI / 180);
int x = (int) (getSpeed() * Math.cos(radian));
int y = (int) (getSpeed() * Math.sin(radian));
changeY(-y);
changeX(x);
if(getPower()!=1){
    Wall temp;
    Iterator<Wall> it = Walls.listIterator();
    while (it.hasNext()){
        temp = it.next();
        if (temp.collision(getX(), getY(), getWidth(), getHeight()))
        {
            changeY(y);
            changeX(-x);
        }
    }
}
```

Flickering Minimap:

When I drew the minimap I drew it on top of both player screens. Because the Game was updating so fast it would cause the minimap to continually flicker. To remedy this instead of drawing two subimages, I drew four. I then drew these subimages in such a way that the application had a blank space where nothing was drawn. This blank space is where I drew the minimap.

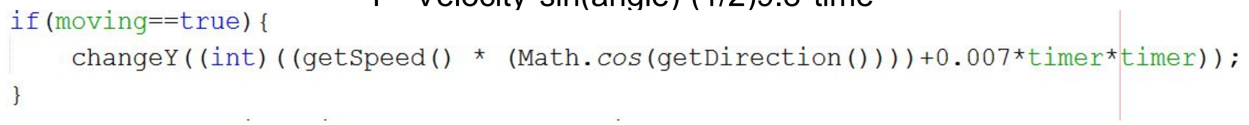

```
Point point=getDrawLoc(player1.get(0));
Image screen1=bimg.getSubimage(point.x, point.y,295,480);
Image screen1_1=bimg.getSubimage(point.x, point.y,295,312);
Image screen1_2=convertToBuffered(screen1).getSubimage(0, 295,236,168);
g.drawImage(screen1_1, 0, 0, this);
g.drawImage(screen1_2, 0, 295, this);
Point point2=getDrawLoc(player2.get(0));
Image screen2=bimg.getSubimage(point2.x, point2.y,295,480);
Image screen2_1=bimg.getSubimage(point2.x, point2.y,295,312);
Image screen2_2=convertToBuffered(screen2).getSubimage(58, 295,236,168);
g.drawImage(screen2_1, 306, 0, this);
g.drawImage(screen2_2, 364, 295, this);
g.setColor(Color.black);
g.fillRect(295,0,10,312);
g.drawRect(295,0,10,312);
Image mini=bimg.getScaledInstance(128, 102, BufferedImage.SCALE_SMOOTH);
g.drawImage(mini, 236, 312, this);
```

Rainbow Reef:

Implementing Gravity:

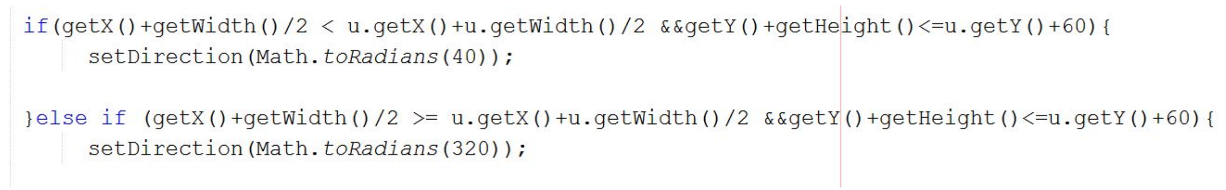
To implement gravity for Pop I used the Y displacement formula as a base for how I changed the Y posotion of Pop. This equation is as follows:

$$Y= Velocity*\sin(angle)-(1/2)9.8*time^2$$


```
if(moving==true){
    changeY((int)((getSpeed() * (Math.cos(getDirection())))+0.007*timer*timer));
}
```

As you can see in my code, the constant (½)*9.8 is replaced with .0007. I found this number by playing around with different numbers. I found that this gravity felt the best for how Pop moves in game.

Changing Direction of Pop on Collision with top of block:


```
if(getX()+getWidth()/2 < u.getX()+u.getWidth()/2 &&getY()+getHeight()<=u.getY()+60){
    setDirection(Math.toRadians(40));

}else if (getX()+getWidth()/2 >= u.getX()+u.getWidth()/2 &&getY()+getHeight()<=u.getY()+60){
    setDirection(Math.toRadians(320));
```

A problem I faced with reflection angles involved when Pop hit the top of a block. When Pop hit the sides of a block I just reversed the angle in which it was moving. However this did not work for the top of the block because Pop would continue to move back and forth rather than continually in one direction. To remedy this I changed Pops movement angle based in where it was when the top of the block was hit. If the middle of pop is to the left of the middle of the block, then pop will move left. Otherwise if it is to the right of the block then it will move to the right.

Reflection:

Overall this project was a great lesson in reusability. The tank game was the first project I worked on and clearly showed. Because I followed Lowelles design pattern many of my classes ended up in the same java file. In fact, the only classes with their own file where the power ups. On top of that I did not make use of reusability. The classes Tank1 and Tank2 were extremely similar, and it would have made more sense to create a single Tank class. My reasoning for two classes was that each Tank used a different type of bullet. This could have been solved by adding a new Bullet argument in the single Tank class constructor.

I’m glad we were tasked to make a second game because I was able to fix my mistakes of the first game. First off, I separated all my different classes into their own java file. Although I could have made different classes for my walls, bricks, and enemies this would have been extremely wasteful and instead I finally made use of reusability. To create the different units all I did was create an instance of the wall class, with different image arrays and different scores.