

Tank Wars Documentation

Albert Du

Project Information:

Tank Wars game

CSC 413 summer 2017 Professor Souza

August 13, 2017

Albert Du

Introduction:

This documentation provides instruction on how to play Tank Wars and how the code functions. Tank Wars is a two-player tank shooter game. This game was written in Java using the Observer design pattern. The program extends JApplet and implements the Runnable interface. Movement is shown by constantly updating and redrawing Observer objects based on Observable game events in Tank Wars. There is one JAR file and one program file: MyTankGame.jar and TankGame.java. Running either file starts the game. MyTankGame.jar can be run from any directory while TankGame.java must be run from an IDE.

Scope of Work:

This project required creating a tank game and another game of choice. The most important aspect of this project was writing reusable code, such as classes and methods which can be reused for another game. Reused TankGame classes are highlighted in the Class Structure

section. Other implementation requirements were smooth performance, sound, tank angle & direction handling, a mini-map, collision detection, explosions, along with health & bonuses. Students were allowed to implement gameplay as they wished.

Background / Resources Provided:

Unlike previous projects in CSC 413, no code was provided to build upon. However, students were provided sprites for the tank game along with source code from a plane game called Airstrike to study. Nearly all the original sprites were .gif files. They were converted to .png images to avoid working with .gifs. Tank Wars borrows much of the class structure from Airstrike. The logic within Airstrike's classes was modified to suit Tank Wars.

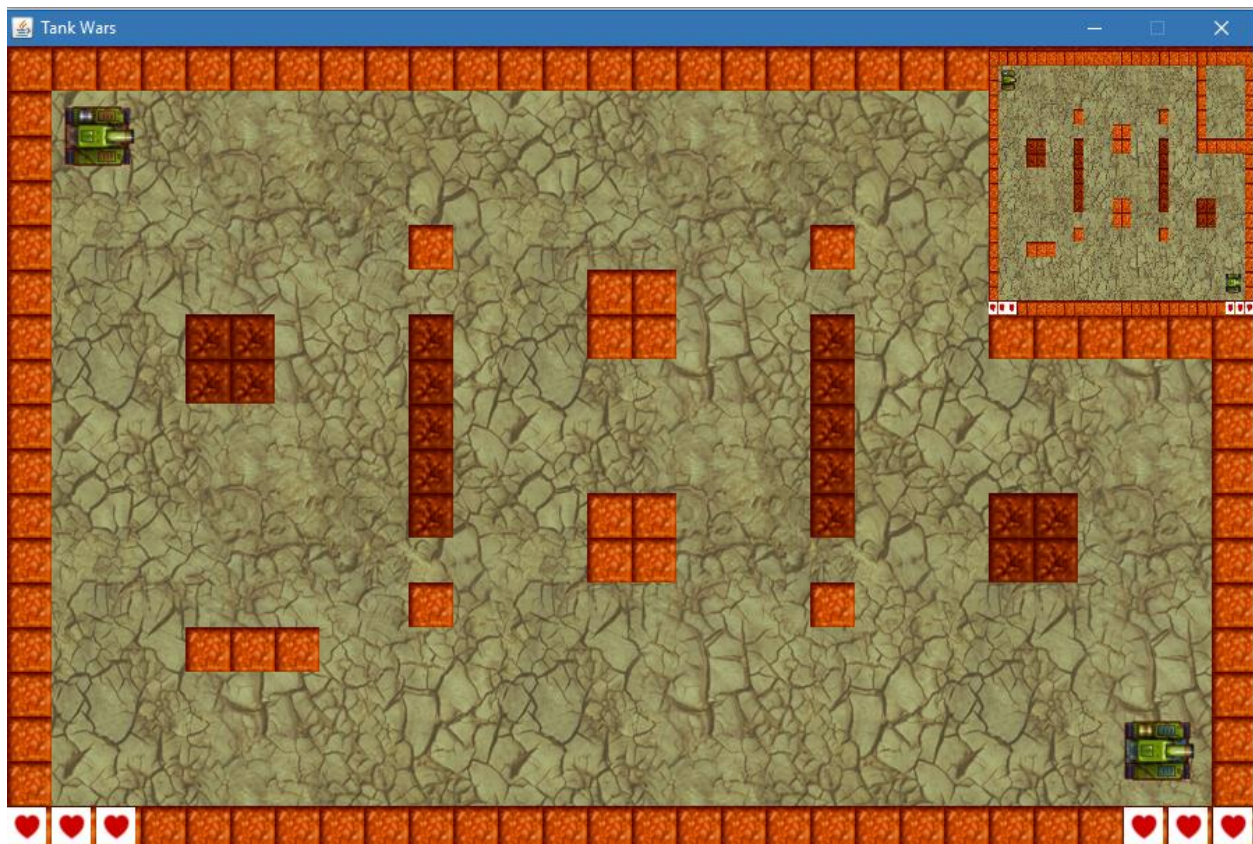
Assumptions:

Tank Wars was written using NetBeans IDE 8.2 on Windows 10. Source code and resources were placed in different folders within the same package. Source code was placed in the tankgame folder while resources were placed in the Resources folder.

How to play Tank Wars:

- Both players start with 3 lives. They are represented by hearts on the bottom corners of the screen.
- Player 1 spawns on the upper left corner of the game map while Player 2 spawns on the lower right corner.
- Player 1 controls: WASD movement, space to fire.

- Player 2 controls: Arrow key movement, enter to fire.
- Power Ups: Green bullet power up and red life power up.
 1. Green bullet power ups allow players to fire a faster traveling green bullet. Players lose the green bullet power up upon death.
 2. Red life power ups give the player one additional life. Only two red life power ups spawn over the duration of the game.
- Upon being hit by a bullet, players explode and lose a life. If the player has at least one life remaining, they will respawn at their spawn point after a short time.
- The game ends once either player is out of lives.



Implementation Details:

Class Structure:

Public class TankGame extends JApplet implements Runnable

Public class CollisionDetector (Detects collisions between Tank Wars objects.)

Public class Explosions (Handles explosions for tanks and walls)

Public class GameEvents extends Observable

Public class KeyControl extends KeyAdapter

Public class Sound

Public class GameObject implements Observer

Public class Bullet extends GameObject

Public class PowerUp extends GameObject

Public class Tank extends GameObject (Parts of this class were reused)

Public class Wall extends GameObject

TankGame serves as the main class for this project. Drawing the game and running is done using a thread in TankGame. The GameObject class stores shared information among all objects in Tank Wars, such as location, size, and speed. Classes extending GameObject are used for specific objects within the game such as bullets, power ups, tanks, and walls. GameObjects observe observable game events in Tank Wars and are updated accordingly.

TankGame class:

Data fields for TankGame.

```
36 import javax.swing.JApplet;
37 import javax.swing.JFrame;
38
39 public class TankGame extends JApplet implements Runnable{
40     private Thread thread;
41     public static GameEvents gameEvent1, gameEvent2;    // Will observe the two players' tanks.
42     public static Tank player1, player2;
43     public int playerID1 = 1, playerID2 = 2, buffTimer = 1;
44     private String winMsg, loseMsg; // Display on game over screen.
45     private boolean cleared;        // Determines if game window cleared before drawing game over screen.
46
47     Image background, tankP1, tankP2, normalBullet, fastBullet, strongWall, weakWall, end,
48         healthBuff;
49     BufferedImage life;
50
51     // Stores explosion animations.
52     static Image[] bigExplosion = new Image[7];
53     static Image[] smallExplosion = new Image[6];
54
55     private BufferedImage bimg;
56     Graphics2D g2;
57
58     boolean gameOver = false;
59     CollisionDetector CD;
60     Sound sp;
61     public InputStream wallMap; // Game map drawn from a text file.
62
63     static ArrayList<Explosions> explosions = new ArrayList<Explosions>(100000);
64     static ArrayList<Wall> wall = new ArrayList<Wall>();
65     static ArrayList<PowerUp> powerUps = new ArrayList<PowerUp>();
66
67     int width = 900, height = 605;
68     private Dimension contentWindowSize; // Real size of the content window.
```

The TankGame class is used to initialize all sprite images within Tank Wars, draw the map, and run the game. The game continues updating and redrawing all game objects using repaint() and drawDemo() until the game is over. Array lists store all active walls, power ups, and bullets in the game.

DrawDemo method:

DrawDemo() continuously draws all game objects in Tank Wars. DrawDemo() checks for collisions, an observable game event being observed by the tanks (along with any game objects associated with the tanks), and spawns power ups according to a timer. DrawDemo() also scales down the image of the game and redraws it in the upper right corner, creating a mini map.

Code for observing game events and collisions:

```
gameEvent1.addObserver(player1);  
gameEvent2.addObserver(player2);  
  
CD = new CollisionDetector(gameEvent1, gameEvent2); // In init()  
  
// Collision detection between game objects. In drawDemo()  
  
CD.playerVSPlayer(player1, player2);  
CD.playerVSBullet(player1, player2);  
CD.playerVSWall(player1, player2);  
CD.playerVSPowerUp(player1, player2);  
CD.bulletVSWall(player1, player2);
```

MapWalls method:

MapWalls() reads in a text file called WallMap.txt from the Resources folder.

WallMap.txt contains only zeroes, ones, and twos. MapWalls() reads each character in WallMap.txt and draws a wall at a specified location depending on the number.

0 – Draw nothing

1 – Draw a non-breakable wall

2 – Draw a breakable wall

Since each wall sprite is 32x32, each column or row in WallMap.txt represents a location 32 pixels apart. The dimensions of the content window are 900x548, meaning WallMap.txt needs to have 29 columns and 18 rows. MapWalls() draws walls at $X = (\text{position} \% 28 * 32)$, $Y = (\text{position} / 28) * 32$.

```

342 private void mapWalls() {
343     BufferedReader line = new BufferedReader(new InputStreamReader(wallMap));
344     String currentLine;
345     int position = 0;
346
347     try {
348         // Reads all characters on each line and map walls accordingly.
349         while ((currentLine = line.readLine()) != null) {
350             for(int i = 0; i < currentLine.length(); i++) {
351                 if(currentLine.charAt(i) == '1') {
352                     wall.add(new Wall(strongWall, (position % 28) * 32, (position / 28) * 32, false);
353                 }
354                 else if(currentLine.charAt(i) == '2') {
355                     wall.add(new Wall(weakWall, (position % 28) * 32, (position / 28) * 32, true));
356                 }
357
358                 position++;
359             }
360         }
361     } catch (Exception e) {
362         System.out.println("Walls: couldn't build");
363     }
364 }
365

```

GameObject class:

GameObject is used to initialize general information about various game objects within Tank Wars such as tanks, walls, bullets, and power ups. Here are the data fields and constructor:

```

public class GameObject implements Observer{
    // Sprite for game object.
    protected Image img;

    // (x,y) coord, size of object, and speed.
    protected int x, y, height, width, speed;

    // Game object exploding.
    protected boolean boom;
    private boolean show;

    public GameObject(Image img, int x, int y, int speed) {
        this.img = img;
        this.x = x;
        this.y = y;
        this.speed = speed;
        this.width = img.getWidth(null);
        this.height = img.getHeight(null);
    }
}

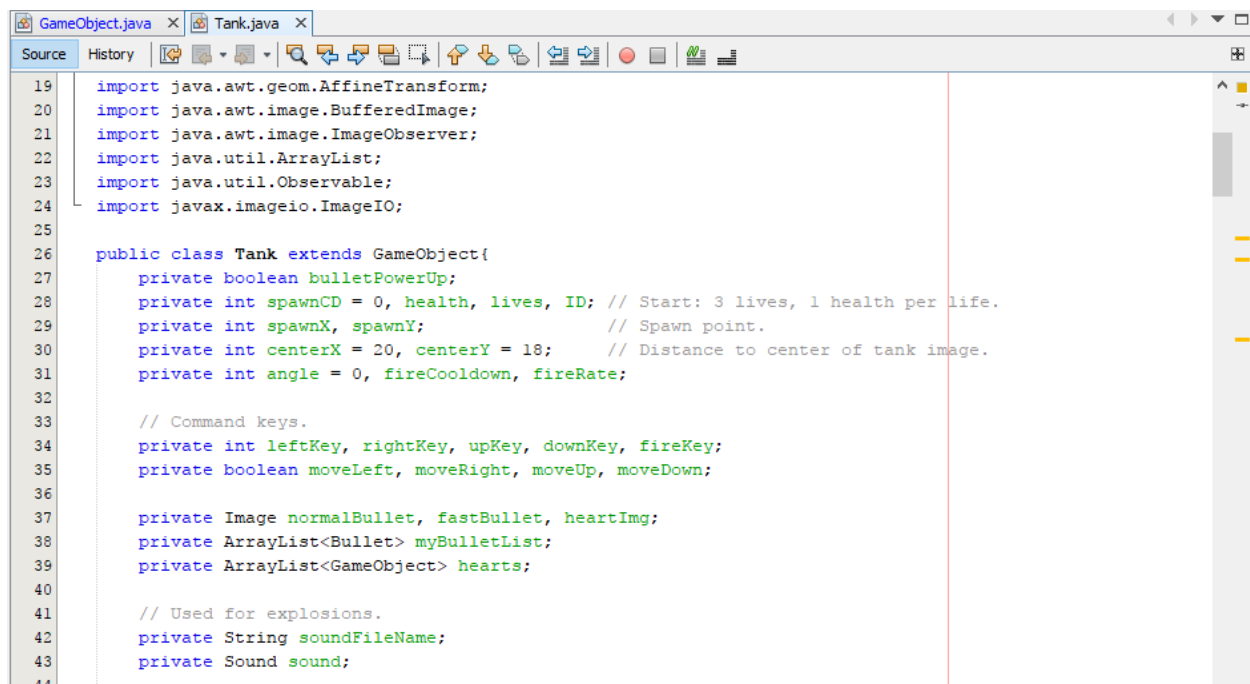
```

All game objects in Tank Wars have an xy-coordinate, a sprite image, speed, and size. They can also explode or become invisible. All game objects will have this information. Therefore, specific game objects will extend the GameObject class. GameObject implements Observer,

therefore it also includes a draw() and update() method. It also includes getters for the data fields. There are only two setters: for the xy-coordinates. Getters and setters for show and boom weren't included because it was forgotten that they are data fields. Instead, show is used as a data field for the PowerUp and Bullet classes. Boom is used as a data field in the Tank class. This error went unnoticed until after the due date.

Tank Class:

Tank data fields:



```
19 import java.awt.geom.AffineTransform;
20 import java.awt.image.BufferedImage;
21 import java.awt.image.ImageObserver;
22 import java.util.ArrayList;
23 import java.util.Observable;
24 import javax.imageio.ImageIO;
25
26 public class Tank extends GameObject{
27     private boolean bulletPowerUp;
28     private int spawnCD = 0, health, lives, ID; // Start: 3 lives, 1 health per life.
29     private int spawnX, spawnY; // Spawn point.
30     private int centerX = 20, centerY = 18; // Distance to center of tank image.
31     private int angle = 0, fireCooldown, fireRate;
32
33     // Command keys.
34     private int leftKey, rightKey, upKey, downKey, fireKey;
35     private boolean moveLeft, moveRight, moveUp, moveDown;
36
37     private Image normalBullet, fastBullet, heartImg;
38     private ArrayList<Bullet> myBulletList;
39     private ArrayList<GameObject> hearts;
40
41     // Used for explosions.
42     private String soundFileName;
43     private Sound sound;
44 }
```

Tank extends GameObject. CenterX and centerY represent the distance from the upper left corner of the tank image to the center: 20 pixels to the right and 18 pixels down. This is used to calculate the offset from the tank's center to the tank's turret. The offset is required to ensure that bullets fire from the tank's turret regardless of which direction the tank is facing. This is further explained in the Bullet class section.

Tank Constructor:

```
Tank.java x Bullet.java x
Source History
43 private Sound sound;
44
45 public Tank(Image img, int lives, int x, int y, int speed, int left, int right,
46 int up, int down, int fire, int ID) {
47     super(img, x, y, speed);
48     this.ID = ID;
49     this.health = 1;
50     this.lives = lives;
51     this.fireRate = 65;
52
53     boom = false;
54     bulletPowerUp = false;
55
56     this.leftKey = left;
57     this.rightKey = right;
58     this.upKey = up;
59     this.downKey = down;
60     this.fireKey = fire;
61
62     this.spawnX = x;
63     this.spawnY = y;
64
65     this.myBulletList = new ArrayList<Bullet>();
66     this.hearts = new ArrayList<GameObject>();
67     this.soundFileName = "Resources/Explosion_large.wav";
68     this.sound = new Sound(2, soundFileName);
69
70     try {
71         normalBullet = ImageIO.read(this.getClass().getClassLoader().getResource(("Resources/Rocket.png")));
72         fastBullet = ImageIO.read(this.getClass().getClassLoader().getResource("Resources/FastBullet.png"));
73         heartImg = ImageIO.read(this.getClass().getClassLoader().getResource(("Resources/Life.png")));
74     }
75     catch (Exception e) {
76         System.out.println("Tank class: Resource not found");
77     }
78 }
```

The constructor initializes each Tank object with 3 lives (represented by hearts), its movement and fire keys (using key listeners), location, sounds, and bullets. Bullets and hearts are placed inside an array list. The bullet list is initially empty and adds a bullet only when the fire key is pressed. When the fire key is pressed, the fire() method is called to add the appropriate bullet to the bullet list. If a tank has a green bullet power up, green bullets are drawn instead of the normal, red bullet. Once a bullet collides with a wall or tank, it's removed from the bullet list.

```
167 public void fire() {
168     Bullet b;
169
170     // Adds x, y offset to the tank's center to ensure
171     // bullet fires from the tank's turret from all angles.
172     // Normal bullets have 7 speed. Fast bullets have 10 speed.
173     if(bulletPowerUp == true) {
174         b = new Bullet(fastBullet, x + centerX, y + centerY, angle, 10);
175         myBulletList.add(b);
176     }
177     else {
178         b = new Bullet(normalBullet, x + centerX, y + centerY, angle, 7);
179         myBulletList.add(b);
180     }
181 }
```

Tank rotation and movement (Draw(), update(), and updatePosition() methods)

The draw() method draws the tank depending on its status in the game. Tanks are only drawn if they're alive. If alive, the tank's position is updated by the update() and updatePosition() methods. Key listeners were added to the tanks in the TankGame class. Since the tanks in Tank Wars are observing gameEvent1 and gameEvent2, they respond to keyboard input to move via the update() method. UpdatePosition() repositions the tank depending on which movement keys were pressed.

```
// Key listener for the two tanks.
```

```
KeyControl key1 = new KeyControl(gameEvent1);
```

```
KeyControl key2 = new KeyControl(gameEvent2);
```

```
addKeyListener(key1);
```

```
addKeyListener(key2);
```

Code for updatePosition()

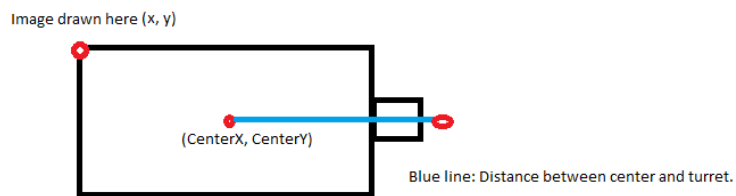
```
301 *
302 * Cited from: http://www.java-gaming.org/index.php?topic=29407.0
303 * Modified to work for Tank Wars.
304 */
305 public void updatePosition(){
306     // Rotates tank left (counter-clockwise).
307     if (moveLeft == true) {
308         angle += 1;
309     }
310
311     // Rotates tank right (clockwise).
312     if (moveRight == true) {
313         angle -= 1;
314     }
315
316     if (moveUp == true) {
317         x += speed * Math.cos(Math.toRadians(6 * angle));
318         y -= speed * Math.sin(Math.toRadians(6 * angle));
319     }
320
321     if (moveDown == true) {
322         x -= speed * Math.cos(Math.toRadians(6 * angle));
323         y += speed * Math.sin(Math.toRadians(6 * angle));
324     }
325
326     // 0 <= (6 * angle) <= 360.
327     if (angle == -1)
328         angle = 59;
329     else if (angle == 60)
330         angle = 0;
331 }
332 }
333
```

An explosion is drawn and an explosion sound is played if the tank has been hit by a bullet.

Bullet Class:

Bullet extends GameObject. The Bullet class handles drawing bullets in Tank Wars. It stores the bullet's angle, speed, and visibility, along with information from the GameObject superclass. When a tank fires a bullet, a bullet image is drawn in front of the tank's turret. Bullets move in a straight line along the direction it was fired from. Initially drawing a bullet requires the

offset between the tank's center and the tank's turret. Since Java draws images from the upper left corner, the distance between a tank's (x, y) coordinate and center must be calculated first. Only then can the offset between center and turret be calculated. With the offset, calculations can be made to determine where a bullet should be drawn. This method ensures that bullets are drawn in front of a tank's turret regardless of the direction (angle) it's facing.



The formula used to calculate the bullet location is:

$$\text{Bullet X} = \text{Center X} + (\text{OffsetX} * \cos(\text{Tank angle}) - (\text{OffsetY} * \sin(\text{Tank angle}))$$

$$\text{Bullet Y} = \text{Center Y} + (\text{OffsetX} * \sin(\text{Tank angle}) - (\text{OffsetY} * \cos(\text{Tank angle}))$$

Implementation of the formula:

CenterX = 20, CenterY = 18, OffsetX = 32, OffsetY = 0, tank angle = bullet angle

```
44 public void update() {
45     x += speed * Math.cos(Math.toRadians(6 * angle));
46     y -= speed * Math.sin(Math.toRadians(6 * angle));
47 }
48
49 @Override
50 public void draw(ImageObserver obs, Graphics2D g2d) {
51     if(show){
52         BufferedImage bimg = (BufferedImage) img;
53         double rotationRequired = Math.toRadians(-6 * angle);
54
55         // newX = (x + (offsetX * Math.cos(angle) - offsetY * Math.sin(angle)));
56         // newY = (y + (offsetX * Math.sin(angle) + offsetY * Math.cos(angle)));
57         int newX = (int) (x + (32 * Math.cos(rotationRequired)) - (0 * Math.sin(rotationRequired)));
58         int newY = (int) (y + (32 * Math.sin(rotationRequired)) + (0 * Math.cos(rotationRequired)));
59
60         // Credit to JavaTutorials101 on YouTube for rotation algorithm.
61         // https://www.youtube.com/watch?v=vHfGiTFWoc4&t=222s
62         AffineTransform at = AffineTransform.getTranslateInstance(newX, newY);
63         at.rotate(rotationRequired, bimg.getWidth() / 2, bimg.getHeight() / 2);
64
65         g2d.drawImage(bimg, at, obs);
66     }
67 }
```

PowerUp Class:

The PowerUp class extends GameObject. The only additional data field is an integer representing power up type. Green bullet power ups are type 1, while red life power ups are type 2. PowerUps do not move, therefore the update() method has no code. PowerUp objects are stored in the powerUps array list in the TankGame class. They're removed from the array list upon colliding with (being picked up by) a tank.

PowerUp class constructor:

```
19  import java.awt.Image;
20  import java.awt.image.ImageObserver;
21
22  public class PowerUp extends GameObject{
23      boolean show = false;
24      int type;
25
26      public PowerUp(Image img, int x, int y, int speed, int type) {
27          super(img, x, y, speed);
28          this.show = true;
29          this.type = type;
30      }
31
```

Wall class:

Wall extends GameObject. Walls are either breakable or not breakable. All walls explode with sound effects when hit by bullets. Wall data fields and constructor:

```
10  L  */
11      package tankgame;
12
13  L  import java.awt.Graphics2D;
14  L  import java.awt.Image;
15  L  import java.awt.image.ImageObserver;
16
17  public class Wall extends GameObject {
18      boolean breakable;
19      private int width, height, respawnCD;    // Breakable walls can respawn.
20      private String soundFileName;           // Explosion sound.
21      Sound sound;
22
23  L  public Wall(Image img, int x, int y, boolean weak) {
24      L      super(img, x, y, 0);
25      L      respawnCD = 0;
26      L      breakable = weak;    // Breakable if a wall is weak.
27      L      this.width = img.getWidth(null);
28      L      this.height = img.getHeight(null);
29      L      boom = false;
30      L      this.soundFileName = "Resources/Explosion_small.wav";
31      L      this.sound = new Sound(2, soundFileName);
32      }
33
```

Height and width could have been removed since GameObject also has a height and width field.

Walls are given an initial respawn cooldown (respawnCD) of 0. If a breakable wall is hit by a bullet, its respawnCD becomes 500 and counts down to 0. The wall respawns once the cooldown reaches 0.

CollisionDetector class:

The CollisionDetector class detects five types of collisions. They are: bullet and wall, player and bullet, player and player, player and power up, and player and wall. Collision detection is done using Rectangles. Rectangles are drawn around the image of game objects. If they intersect, the objects have collided.

Explosion class:

```
public class Explosions {
    private Image[] img;
    private int x, y;      // Coordinates.
    private int count;     // Explosion timer.
    private boolean finished;

    public Explosions(int x, int y, Image[] img){
        this.x = x;
        this.y = y;
        this.count = 0;
        this.finished = false;
        this.img = img;
    }
}
```

The Explosion class handles the drawing of explosions. Explosions were originally .gifs but were converted to .png. There are two types of explosions: big explosions for tanks, and small explosions for walls. Explosions are drawn when a bullet collides with either a tank or wall. The addExplosion() method is called from the Tank or Wall class, which adds the

appropriate explosion to TankGame's explosion array list. In png form, each explosion has seven images, which are stored in separate Image arrays during the init() method of TankGame.

```
bigExplosion[0] = ImageIO.read(this.getClass().getClassLoader().getResource("Resources/Explosion_0.png"));
bigExplosion[1] = ImageIO.read(this.getClass().getClassLoader().getResource("Resources/Explosion_1.png"));
bigExplosion[2] = ImageIO.read(this.getClass().getClassLoader().getResource("Resources/Explosion_2.png"));
bigExplosion[3] = ImageIO.read(this.getClass().getClassLoader().getResource("Resources/Explosion_3.png"));
bigExplosion[4] = ImageIO.read(this.getClass().getClassLoader().getResource("Resources/Explosion_4.png"));
bigExplosion[5] = ImageIO.read(this.getClass().getClassLoader().getResource("Resources/Explosion_5.png"));
bigExplosion[6] = ImageIO.read(this.getClass().getClassLoader().getResource("Resources/Explosion_6.png"));

smallExplosion[0] = ImageIO.read(this.getClass().getClassLoader().getResource("Resources/Explosion_0.png"));
smallExplosion[1] = ImageIO.read(this.getClass().getClassLoader().getResource("Resources/Explosion_1.png"));
smallExplosion[2] = ImageIO.read(this.getClass().getClassLoader().getResource("Resources/Explosion_2.png"));
smallExplosion[3] = ImageIO.read(this.getClass().getClassLoader().getResource("Resources/Explosion_3.png"));
smallExplosion[4] = ImageIO.read(this.getClass().getClassLoader().getResource("Resources/Explosion_4.png"));
smallExplosion[5] = ImageIO.read(this.getClass().getClassLoader().getResource("Resources/Explosion_5.png"));
```

Explosions are drawn by iterating through each Image in the explosion array, drawing each explosion frame. Once each frame has been drawn, the explosion is removed from the explosion array list.

Sound Class:

The Sound class handles music and explosion sounds in Tank Wars. Music loops infinitely while explosion sounds play only once. Sound objects are created by reading a .mp3, .wav, or .mid file from the Resources folder.

GameEvents Class:

GameEvents handles observable game events in Tank Wars. It's used to add tanks as observers to game events such as collisions.

KeyControl Class:

The KeyControl class extends KeyAdapter. This is to avoid implementing the entire KeyListener interface. KeyControl is used to add key listeners to the tanks. In KeyControl, only the keyPressed() and keyReleased() methods are overridden.

Reusability:

Seven (and parts of an eighth) classes from Tank Wars were reusable for the second game project: Super Rainbow Reef. GameEvents, KeyControl, and Sound were reused line for line. Other classes required modified logic to work for Super Rainbow Reef, but the method structure stayed mostly intact. All reused classes are highlighted in the Class Structure section.

Reflection:

Overall, this project was very challenging and time-consuming. However, I learned much about software development and how much work it takes to make even a small game. I had no prior experience making a game and didn't know where to begin. Studying the code from Airstrike helped tremendously, as well as searching Stack Overflow for similar problems whenever I was stuck. The biggest lesson I learned from making Tank Wars is the importance of inheritance and writing reusable code. It saved so much work for my second game. I am more confident about my software development skills due to completing this project.

