Tank Game And Lazarus

Professor: Anthony J Souza

Developers: Richa Mirashi

Saengduean Calderaz

Jirat Parkeenvincha

Developer's Guide

Contents

Project Information……………………………………………………………………..4
Introduction……………………………………………………………………………..4
Scope Of Work…………………………………………………………………….4
Background/Given Resources…………………………………………………………..5
Assumption……………………………………………………………………………..5
How to Run The Project………………….………………………………………….....5
Implementation Details……………………………………………………………......6
Tank Game ……………....………………………………………………………………6
Tank Game Class Diagram……………………………………………………………7
Class Overview………………………………………………………………......7
Commons package…………………………………………………………………...7
AudioPlayer Class…………………………………………………………...7
CommonAPIs Class……………………………………………………………8
Globals Class……………………………………………………………………..8
MapReader Class………………………………………………………………8
TankOrientation Class……………………………………………………8
Component package……………………………………………………………8
TankObject Class………………………………………………………………8
CollisionDetector Class…………………………………………………..9
KeysControl Class………………………………………………………10
Bullet Class……………………………………………………………………...12
Explosion Class…………………………………………………………………13
Core package………………………………………………………………………..13
Driver Class……………………………………………………………………..13
TankWorld Class……………………………………………………...14
Lazarus game…………………………………………………….........................19
Lazarus Game Class Diagram…………………………………………………...20
Class Overview………………………………………………………………....20
Common Package………………………………………………………………..20
AudioPlayer Class………………………………………………………………..20
CommonAPIs Class…………………………………………………………...21
Globals Class……………………………………………………………………21
MapReader Class……………………………………………………………...21
SpawnBoxes Class……………………………………………………………..21
Component Package…………………………………………………………...22
Animation Class……………………………………………………………… 23
Box Class……………………………………………………………………...24
CollisionDetector Class…………………………………………………….25

KeysControl Class……………………………………………………………….26
Lazarus Class…………………………………………………………………….28
Core Package……………………………………………………………………….28
Driver Class……………………………………………………………………...28
LazarusWorld Class…………………………………………………………...29
Chart of classes shared between two games……………………………………………..35
Self-Reflection on Software Design, Development and Reusability…………………………….35
Conclusion……………………………………………………………………………….36
Future Scope……………………………………………………………………………..36

Project Information:

CSC 413 Summer 2017	
Team number:	Team02
Team Projects:	**Tank Game and Lazarus Game**
Links to repository:	
Tank Game:	https://github.com/CSC413-SFSU/csc413-TankGame-Team02
Lazarus:	https://github.com/CSC413-SFSU/csc413-lazarus-Team02

Introduction:

For this project, we had formed a group of 3 students and tasked to develop two games using JAVA. Our tasks were to design a Tank game based on the information and resources provided by the instructor. Then, we implemented the Tank game based on our design plan and followed the specifications given by the instructor. When the Tank game was partially completed, we had given a task to choose the second game to implement and reuse the classes and code from the Tank game as much as we can.

Tank game is a two players game where they fight against each other. The tanks can move left, right, up and down. There are two kinds of walls, breakable and unbreakable wall. Each tank has 3 lives and their health reduces after getting a shot at. If one tank runs out of lives, then the game will be over and a screen will pop up, displaying who has won the game and asks for the user input if the user wants to play or exit the game.

Lazarus is the second game that our group has implemented. We chose Lazarus game because of its similarities and classes reusability of the tank game. Lazarus is one player game and the player has to help Lazarus escapes from its abductor. Lazarus can move left, right, can jump left and right. The Lazarus will try to create the stairs by using the falling boxes and if it get to the stop sign, it will move to the next level. The game has two levels and each level has different map and difficulty. There are four kinds of boxes and they differ in strength: metal, stone, wood, and cardboard in descending order. If the stronger box falls on top of the lighter one, the lighter one will disappear. The Lazarus will die if any kind of box falls on top of it.

Scope Of Work:

We had to build the Tank game project from the scratch. Thus, we had to design the program structure and had to determine which classes to use to build the program. Since this project is a team project, we had to divide the responsibilities and make sure that each one of us doesn't work on the same task. The Tank game involved the implementation of the main design map, tanks' movement, breakable and unbreakable wall, the collision detector, bullets, explosion, health, lives, sound, minimap and game restart sections. The Lazarus game involved the implementation of the main design map, Lazarus movement, four types of boxes, the collision detector, game reset, game level and sound sections.

Background/Given Resources:

The instructor provided several resources such as the Gamemaker for the Tank and the Airplane game. He also provided us the resources (images and sound) for both the games. In addition, there was no restriction on how to design or how to implement the game, but we had to complete the specifications based on the milestone given by the instructor.

Assumptions:

IntelliJ IDEA Community edition 2017 using Java SE Runtime environment 8 build 1.8.0. was used as a development environment for both the projects. The following assumptions were made while developing the project:

For Tank game:

1. The breakable wall is not regenerated after getting broken up with the bullets.
2. Tanks move only in four directions, Left, Right, Up and Down.
3. Tank's health reduces after getting shot by its own bullet.

For Lazarus game:

1. Lazarus moves only by one block.
2. There are only two levels.

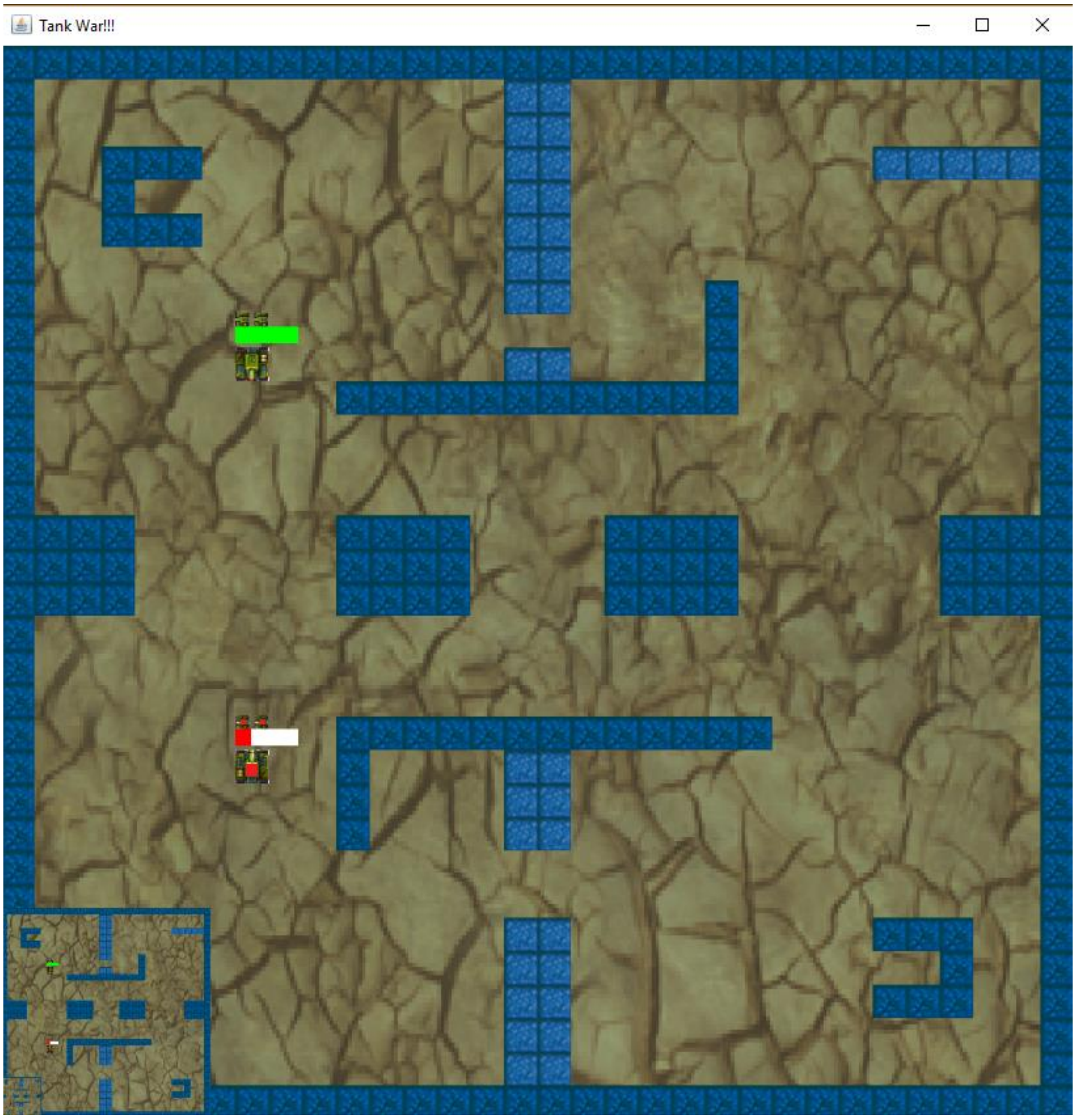
How To Run The Project:

The procedure for running both the projects is same and is as below:

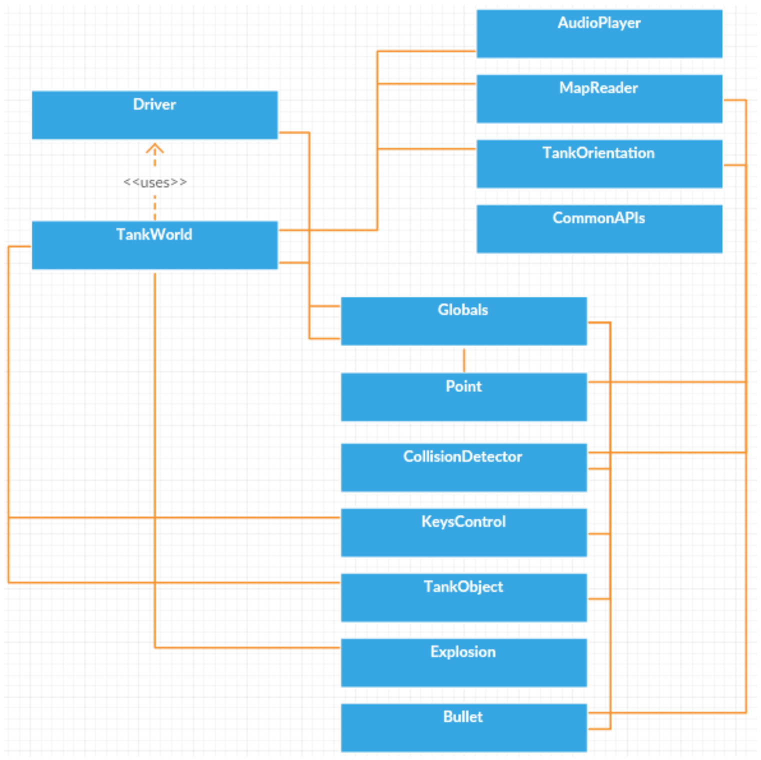
1. Clone the project from the git repository links.
2. Open IntelliJ and open the project.
3. Run the Driver class (SetupSDK 1.8 if required).

Implementation Details:

Tank Game:



Tank Game Class Diagram:



Class Overview:

The Tank game is divided into three packages: commons, components and core.

commons package:

AudioPlayer Class
+Audioplayer +Audioplayer(TankWorld wingman, String filename) +play(): void +loop():void +stop():void

The AudioPlayer class is responsible for loading, playing and looping sound file within the game.

CommonAPIs Class
+validateFileExits(String filename):void +loadImages(String path): BufferedImage

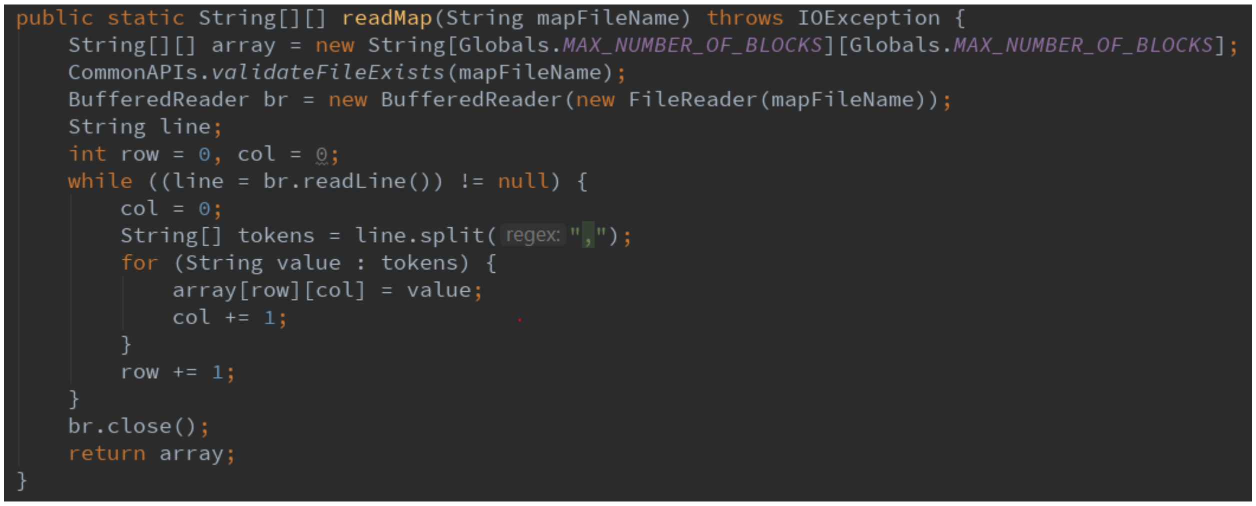
The CommonAPIs class contains all the common functions used all over the project.

Globals Class:

This class has all the global variables that are used all over the project.

MapReader Class
+readMap(String mapFileName): String

MapReader class reads the map1.csv file and creates the map of the Tank game.



```java
public static String[][] readMap(String mapFileName) throws IOException {
    String[][] array = new String[Globals.MAX_NUMBER_OF_BLOCKS][Globals.MAX_NUMBER_OF_BLOCKS];
    CommonAPIs.validateFileExists(mapFileName);
    BufferedReader br = new BufferedReader(new FileReader(mapFileName));
    String line;
    int row = 0, col = 0;
    while ((line = br.readLine()) != null) {
        col = 0;
        String[] tokens = line.split( regex: ",");
        for (String value : tokens) {
            array[row][col] = value;
            col += 1;
        }
        row += 1;
    }
    br.close();
    return array;
}
```

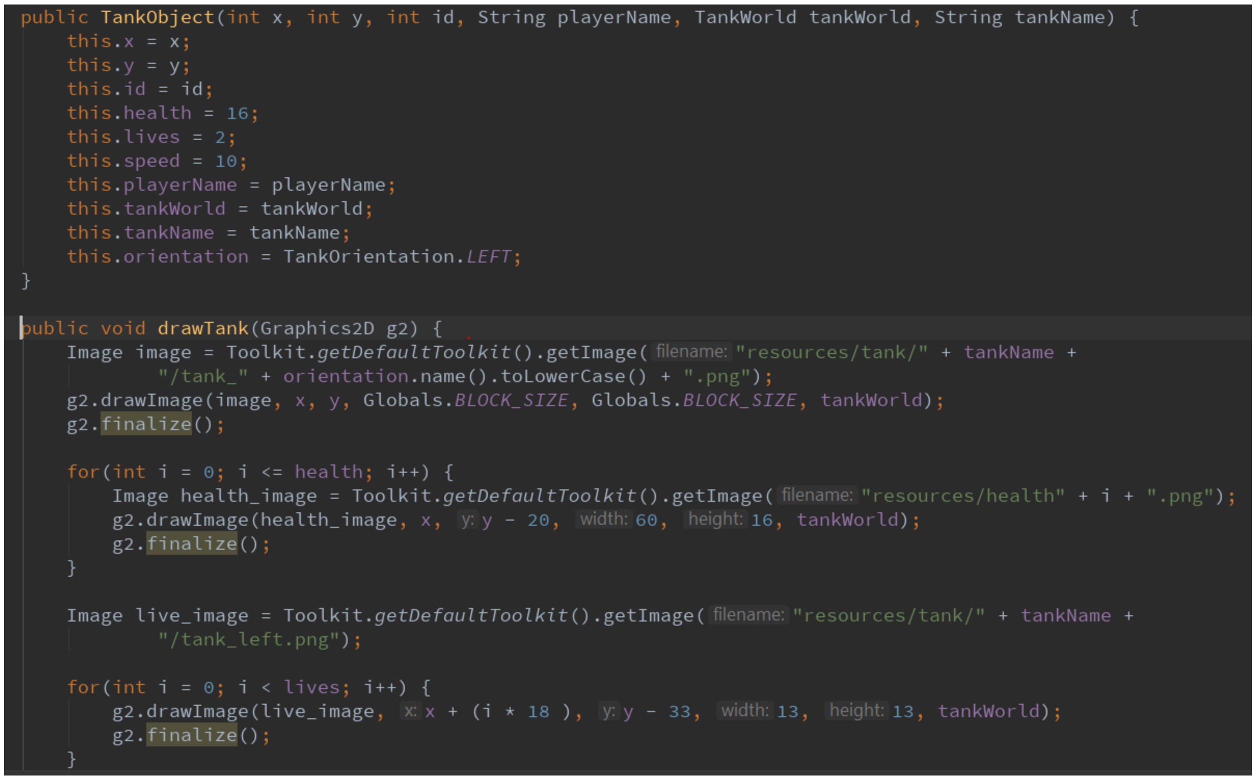
TankOrientation Class:

This is an enum class that has all the tank orientations in it.

components package:

TankObject Class
+TankObject(int x, int y, int id, String playername, TankWorld tankWorld, String tankName) +drawTank(Graphics2D g2): void

TankObject class contains all the information about tank object such as the position x, y, orientation of tank, health, lives, and the name of the tank. This class also draws tank on the map, and displays its health and lives that will appear above the tank.



```
public TankObject(int x, int y, int id, String playerName, TankWorld tankWorld, String tankName) {
    this.x = x;
    this.y = y;
    this.id = id;
    this.health = 16;
    this.lives = 2;
    this.speed = 10;
    this.playerName = playerName;
    this.tankWorld = tankWorld;
    this.tankName = tankName;
    this.orientation = TankOrientation.LEFT;
}

public void drawTank(Graphics2D g2) {
    Image image = Toolkit.getDefaultToolkit().getImage( filename: "resources/tank/" + tankName +
            "/tank_" + orientation.name().toLowerCase() + ".png");
    g2.drawImage(image, x, y, Globals.BLOCK_SIZE, Globals.BLOCK_SIZE, tankWorld);
    g2.finalize();

    for(int i = 0; i <= health; i++) {
        Image health_image = Toolkit.getDefaultToolkit().getImage( filename: "resources/health" + i + ".png");
        g2.drawImage(health_image, x,  y: y - 20,  width: 60,  height: 16, tankWorld);
        g2.finalize();
    }

    Image live_image = Toolkit.getDefaultToolkit().getImage( filename: "resources/tank/" + tankName +
            "/tank_left.png");

    for(int i = 0; i < lives; i++) {
        g2.drawImage(live_image,  x: x + (i * 18 ),  y: y - 33,  width: 13,  height: 13, tankWorld);
        g2.finalize();
    }
```

CollisionDetector Class
+collisionDetector(String [][] map) +validateCollision(int newX, int newY, TankOject otherTank): boolean +validateTankCollision(int newX, int newY, TankObject otherTank): boolean +validateMapCollision(int newX, int newY): boolean +validateBulletoWalllCollsion(Bullet bullet): boolean +validateBullettoTankCollsion(Bullet bullet, TankObject tank1, TankObject tank2): boolean +isGameOver(): boolean +getTankWon(): TankObject
-validateBullettoTankCollision(Bullet bullet, TankObject tank)

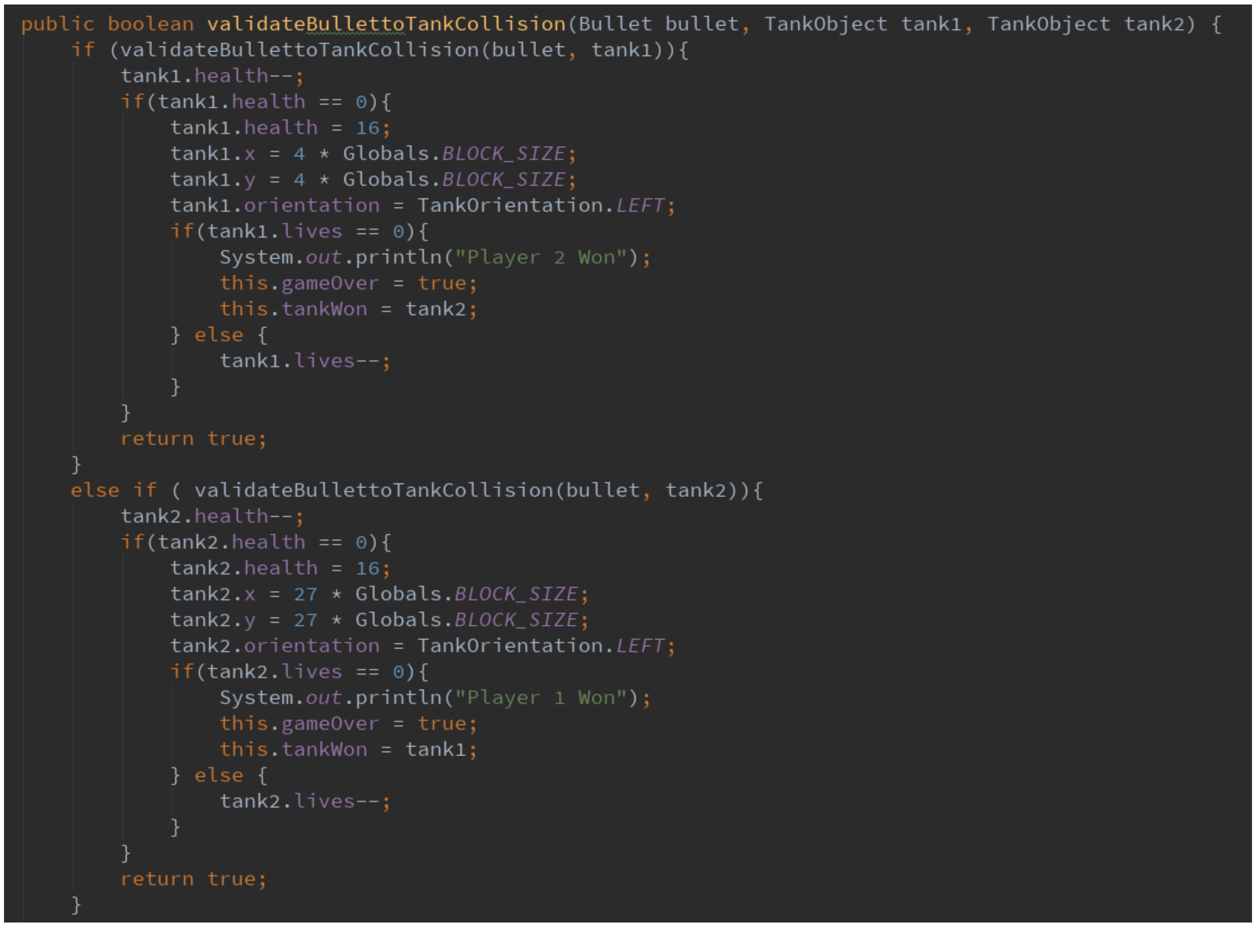
CollisionDector class checks if there is any collision between the tank to wall, bullet to wall, bullet to tank, and tank to another tank. The tanks can not cross the wall or boundary, and even each other. If the bullets collide with the tank, the bullets will damage the tanks. The isGameOver() method checks if the game is over or not. If the game is over, getTankWon() method will determine which tank has won the game.

validateTankCollision(): validates tank to another tank collision

validateMapCollision(): validates tank to unbreakable and breakable wall collision

validateBullettoWallCollision(): gets bullet's orientation and its next position. If the next position is equal to the unbreakable wall, it returns true. If the next position is equal to the breakable wall, it removes the breakable wall from the map and returns true.

validateBullettoTankCollision(): checks if the bullet collides with tank1 or tank2, and accordingly reduces tank's health, lives, and returns true.



```
public boolean validateBullettoTankCollision(Bullet bullet, TankObject tank1, TankObject tank2) {
    if (validateBullettoTankCollision(bullet, tank1)){
        tank1.health--;
        if(tank1.health == 0){
            tank1.health = 16;
            tank1.x = 4 * Globals.BLOCK_SIZE;
            tank1.y = 4 * Globals.BLOCK_SIZE;
            tank1.orientation = TankOrientation.LEFT;
            if(tank1.lives == 0){
                System.out.println("Player 2 Won");
                this.gameOver = true;
                this.tankWon = tank2;
            } else {
                tank1.lives--;
            }
        }
        return true;
    }
    else if ( validateBullettoTankCollision(bullet, tank2)){
        tank2.health--;
        if(tank2.health == 0){
            tank2.health = 16;
            tank2.x = 27 * Globals.BLOCK_SIZE;
            tank2.y = 27 * Globals.BLOCK_SIZE;
            tank2.orientation = TankOrientation.LEFT;
            if(tank2.lives == 0){
                System.out.println("Player 1 Won");
                this.gameOver = true;
                this.tankWon = tank1;
            } else {
                tank2.lives--;
            }
        }
        return true;
    }
```

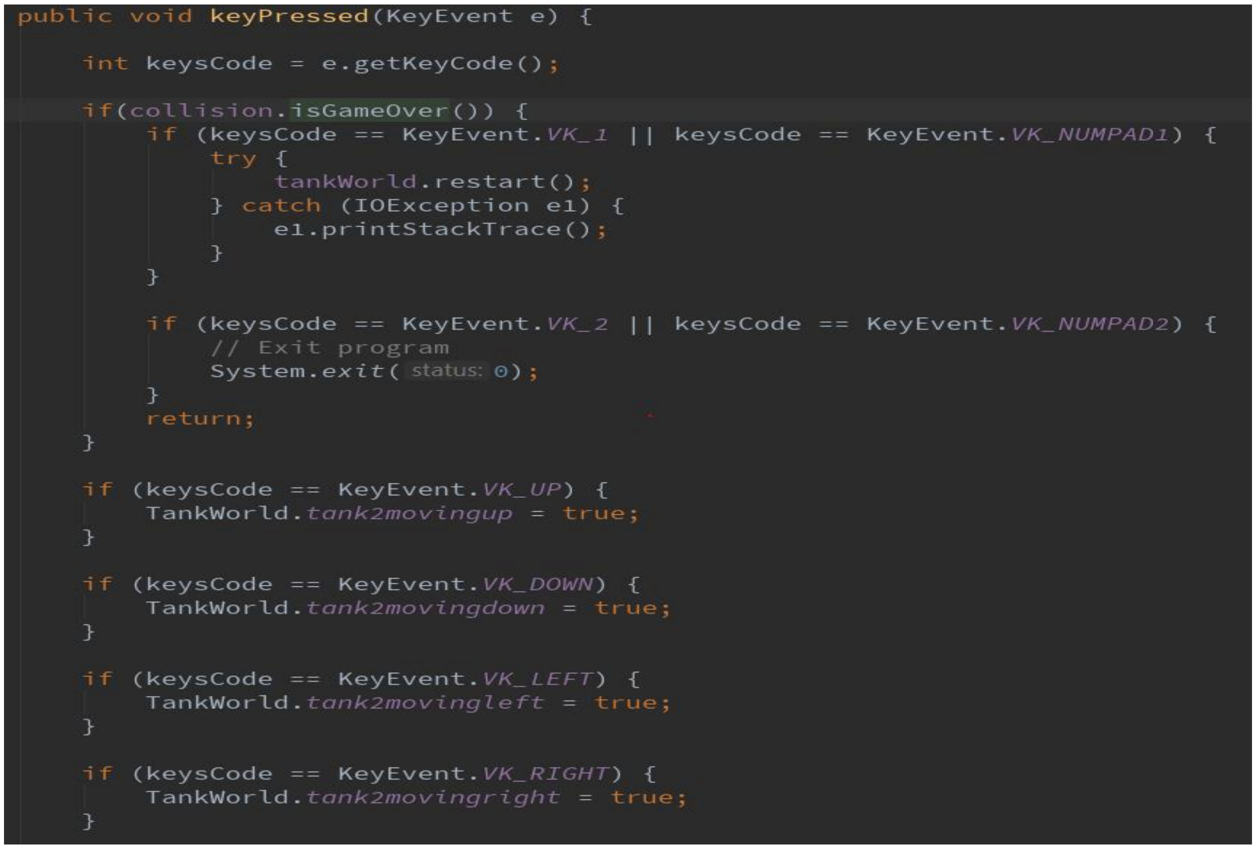
KeysControl Class
+KeysControl(CollisionDectector collisionDetector, TankObject tank1, TankObject tank2, ArrayList<Bullet> bullets, TankWorld tankWorld)
+keyPressed(KeyEvent e): void
+keyReleased(KeyEvent e): void

We have assigned keys to the tank's movement in keyPressed() method. The tanks can move only in four directions: Left, Right, Up and Down.

	Tank 1	Tank 2
Left	A	Left arrow
Right	D	Right arrow
Up	W	Up arrow
Down	S	Down arrow
Fire	Space bar	Numpad0

In addition, this class handles the cases for game restart and game exit.

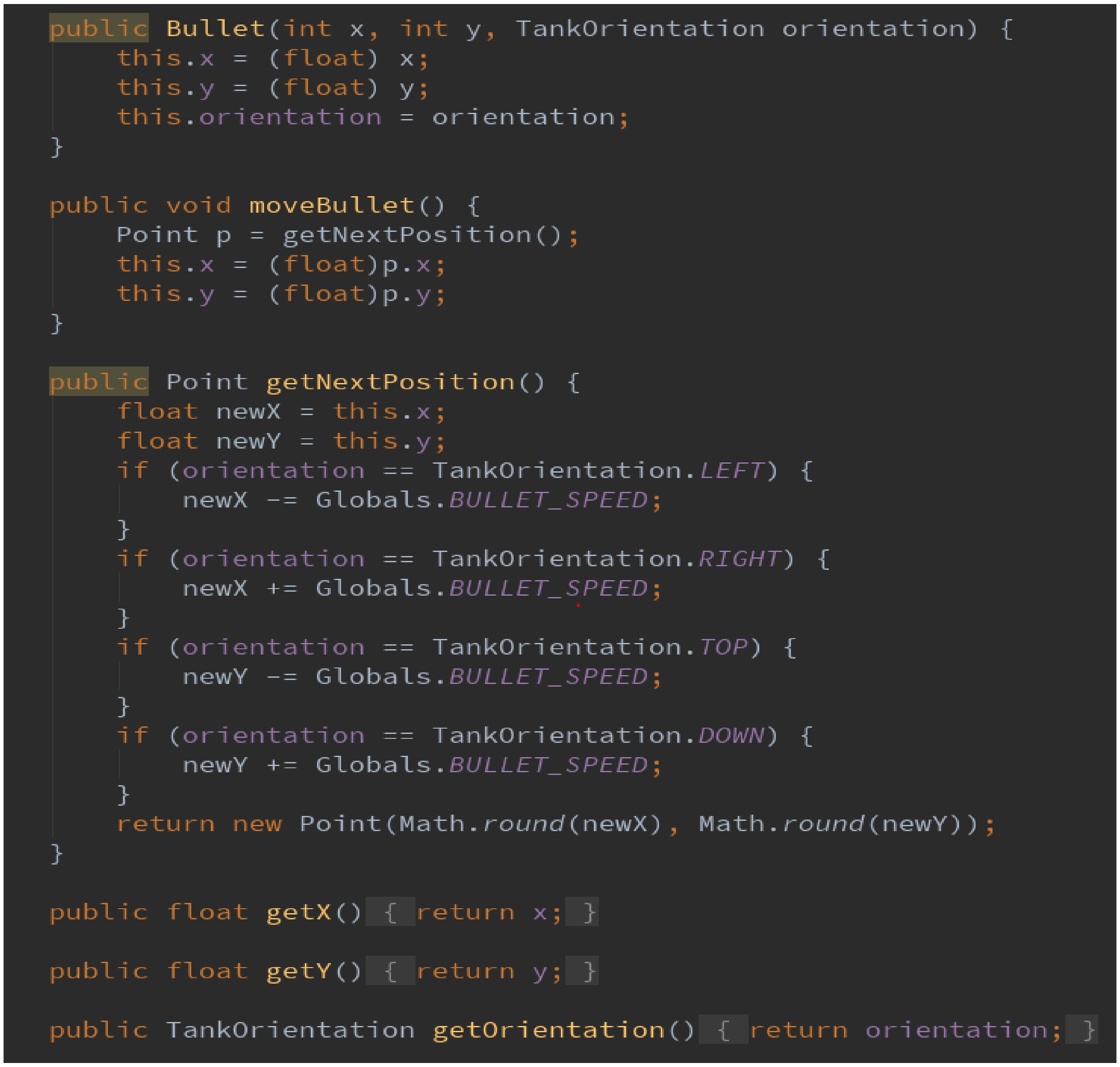
Game restart	1 or Numpad1
Game exit	2 or Numpad2



```
public void keyPressed(KeyEvent e) {

    int keysCode = e.getKeyCode();

    if(collision.isGameOver()) {
        if (keysCode == KeyEvent.VK_1 || keysCode == KeyEvent.VK_NUMPAD1) {
            try {
                tankWorld.restart();
            } catch (IOException e1) {
                e1.printStackTrace();
            }
        }

        if (keysCode == KeyEvent.VK_2 || keysCode == KeyEvent.VK_NUMPAD2) {
            // Exit program
            System.exit( status: 0);
        }
        return;
    }

    if (keysCode == KeyEvent.VK_UP) {
        TankWorld.tank2movingup = true;
    }

    if (keysCode == KeyEvent.VK_DOWN) {
        TankWorld.tank2movingdown = true;
    }

    if (keysCode == KeyEvent.VK_LEFT) {
        TankWorld.tank2movingleft = true;
    }

    if (keysCode == KeyEvent.VK_RIGHT) {
        TankWorld.tank2movingright = true;
    }
```

Bullet Class
+Bullet(int x, int y, TankOrientation orientation) +moveBullet(): void +getNextPosition(): Point +getX(): float +getY():float +getOrientation(): TankOrientation

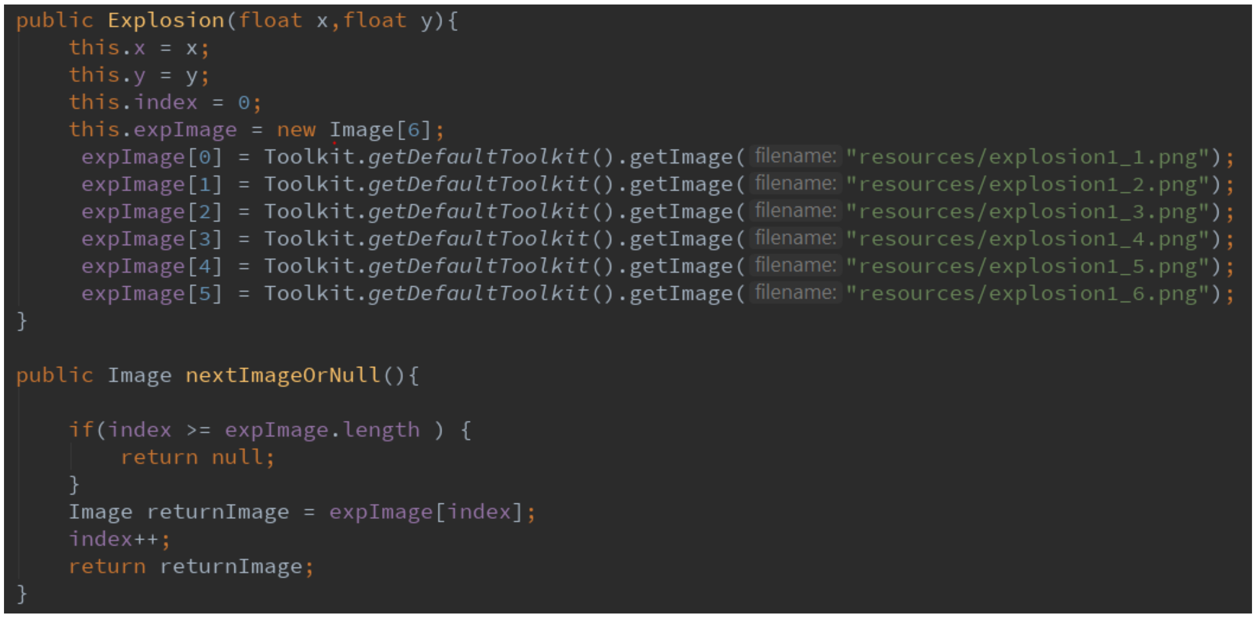
The Bullet class moves the bullet by BULLET_SPEED according to the tank's orientation.



```java
public Bullet(int x, int y, TankOrientation orientation) {
    this.x = (float) x;
    this.y = (float) y;
    this.orientation = orientation;
}

public void moveBullet() {
    Point p = getNextPosition();
    this.x = (float)p.x;
    this.y = (float)p.y;
}

public Point getNextPosition() {
    float newX = this.x;
    float newY = this.y;
    if (orientation == TankOrientation.LEFT) {
        newX -= Globals.BULLET_SPEED;
    }
    if (orientation == TankOrientation.RIGHT) {
        newX += Globals.BULLET_SPEED;
    }
    if (orientation == TankOrientation.TOP) {
        newY -= Globals.BULLET_SPEED;
    }
    if (orientation == TankOrientation.DOWN) {
        newY += Globals.BULLET_SPEED;
    }
    return new Point(Math.round(newX), Math.round(newY));
}

public float getX() { return x; }

public float getY() { return y; }

public TankOrientation getOrientation() { return orientation; }
```

Explosion Class
+Explosion(float x, float y) +nextImageOrNull(): Image +getX(): float +getY(): float

Explosion class loads the explosion images first and then puts these images into an array. The nextImageOrNull() method checks for the next image to animate that happens in the TankWorld class. If there is no image or if the index of images are greater than the array length, this method will return null.

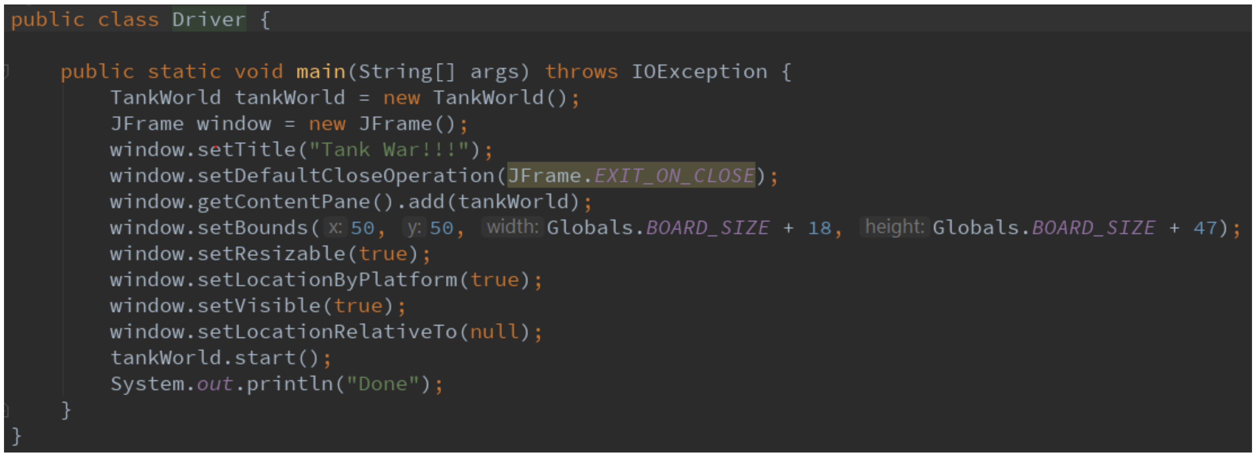


```java
public Explosion(float x,float y){
    this.x = x;
    this.y = y;
    this.index = 0;
    this.expImage = new Image[6];
     expImage[0] = Toolkit.getDefaultToolkit().getImage( filename: "resources/explosion1_1.png");
     expImage[1] = Toolkit.getDefaultToolkit().getImage( filename: "resources/explosion1_2.png");
     expImage[2] = Toolkit.getDefaultToolkit().getImage( filename: "resources/explosion1_3.png");
     expImage[3] = Toolkit.getDefaultToolkit().getImage( filename: "resources/explosion1_4.png");
     expImage[4] = Toolkit.getDefaultToolkit().getImage( filename: "resources/explosion1_5.png");
     expImage[5] = Toolkit.getDefaultToolkit().getImage( filename: "resources/explosion1_6.png");
}

public Image nextImageOrNull(){

    if(index >= expImage.length ) {
        return null;
    }
    Image returnImage = expImage[index];
    index++;
    return returnImage;
}
```

core package:

Driver class:

This class contains the main() method of the game. It creates the window for this game by using Java swing.

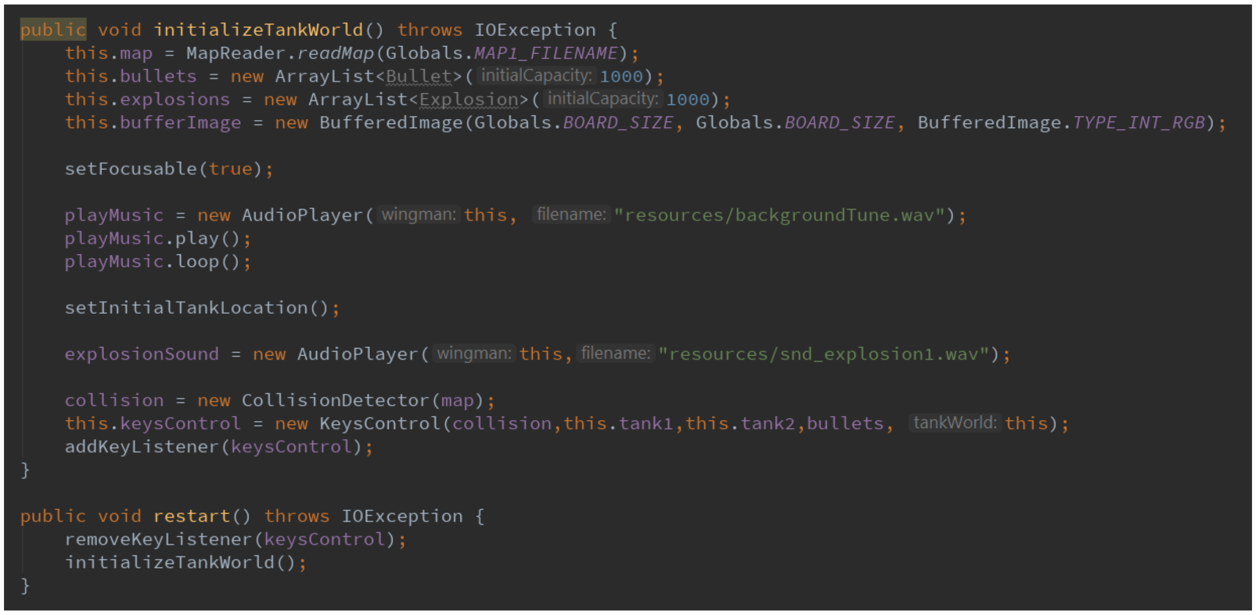


```
public class Driver {

    public static void main(String[] args) throws IOException {
        TankWorld tankWorld = new TankWorld();
        JFrame window = new JFrame();
        window.setTitle("Tank War!!!");
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        window.getContentPane().add(tankWorld);
        window.setBounds( x: 50, y: 50, width: Globals.BOARD_SIZE + 18, height: Globals.BOARD_SIZE + 47);
        window.setResizable(true);
        window.setLocationByPlatform(true);
        window.setVisible(true);
        window.setLocationRelativeTo(null);
        tankWorld.start();
        System.out.println("Done");
    }
}
```

TankWorld Class
+intializaTankWorld(): void +restart(): void +setInitialTankLocation(): void +paint(Graphics g): void +renderMiniMap(Graphic2D g2): void +handleMovement(Graphics g): void +renderGameOver(Graphics g): void +moveBullets(TankOrientation tankOrientation1, TankOrientation tankOrientation2): void +renderExplosion(Graphics g): void +run(): void +start(): void
-renderTankCurrentLocation(Graphics2D g2): void -renderMap(Graphics2D g2): void -renderUnBreakableWall(Graphics2D g2): void -renderBreakableWall(Graphics2D g2): void -renderBackground(Graphics2D g2): void -renderBullets(Graphics g): void

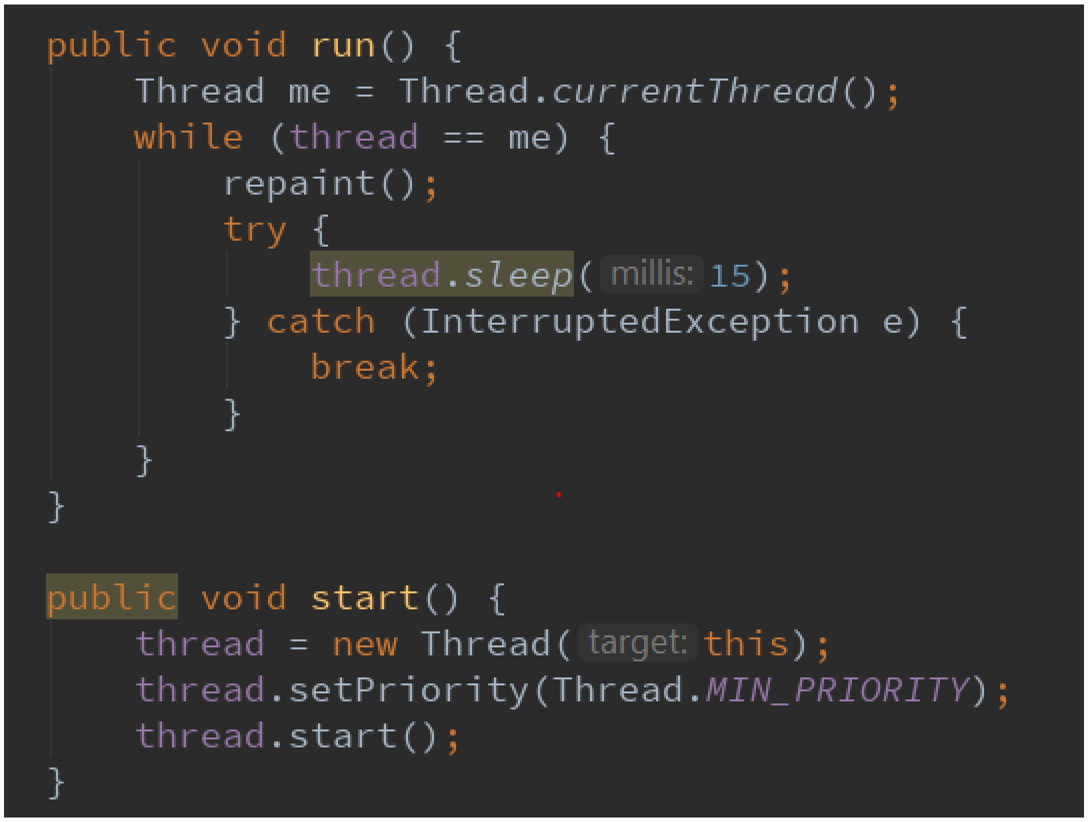
TankWorld is the actual core class of the Tank game, and it implements Runnable interface. This class encapsulates all other classes and paints all the game objects, map, background, minimap, game over screen accordingly on the JFrame window. In addition, the tank's movement, explosion animation and bullets movement are handled in this class after doing the validations.

In the constructor, initializeTankWorld() method is called which initializes map, bullets arraylist, sound, tanks' initial locations, collision and keysControl objects.



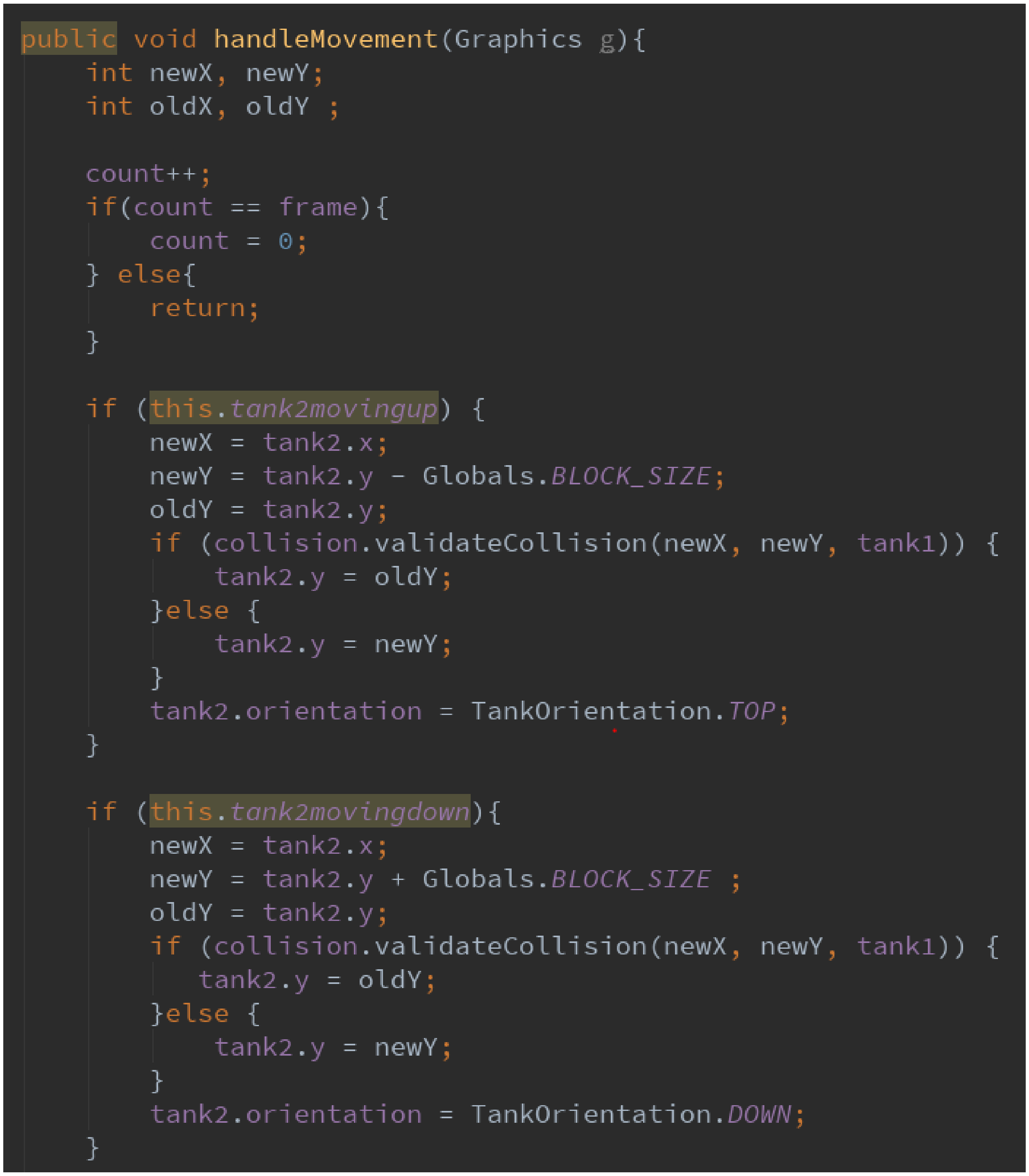
```java
public void initializeTankWorld() throws IOException {
    this.map = MapReader.readMap(Globals.MAP1_FILENAME);
    this.bullets = new ArrayList<Bullet>( initialCapacity: 1000);
    this.explosions = new ArrayList<Explosion>( initialCapacity: 1000);
    this.bufferImage = new BufferedImage(Globals.BOARD_SIZE, Globals.BOARD_SIZE, BufferedImage.TYPE_INT_RGB);

    setFocusable(true);

    playMusic = new AudioPlayer( wingman: this,  filename: "resources/backgroundTune.wav");
    playMusic.play();
    playMusic.loop();

    setInitialTankLocation();

    explosionSound = new AudioPlayer( wingman: this, filename: "resources/snd_explosion1.wav");

    collision = new CollisionDetector(map);
    this.keysControl = new KeysControl(collision,this.tank1,this.tank2,bullets,  tankWorld: this);
    addKeyListener(keysControl);
}

public void restart() throws IOException {
    removeKeyListener(keysControl);
    initializeTankWorld();
}
```

The Driver class calls this start() method in its main() method. The main thread starts here in this start() method. In run() method, repaint() function paints all the game objects, background, minimap etc. on the window screen repeatedly.



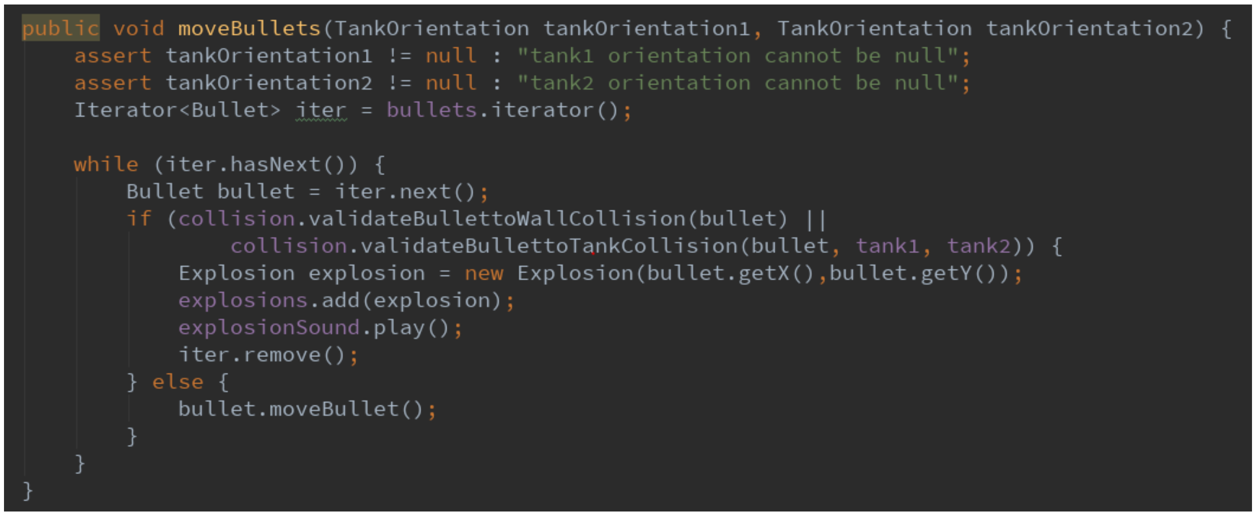
```java
public void run() {
    Thread me = Thread.currentThread();
    while (thread == me) {
        repaint();
        try {
            thread.sleep( millis: 15);
        } catch (InterruptedException e) {
            break;
        }
    }
}

public void start() {
    thread = new Thread( target: this);
    thread.setPriority(Thread.MIN_PRIORITY);
    thread.start();
}
```

handleMovement(): Every time when tank moves, this function checks if there is any collision of tank with the walls or with another tank, and if no, then tank moves to its next position, else stops at its current position.



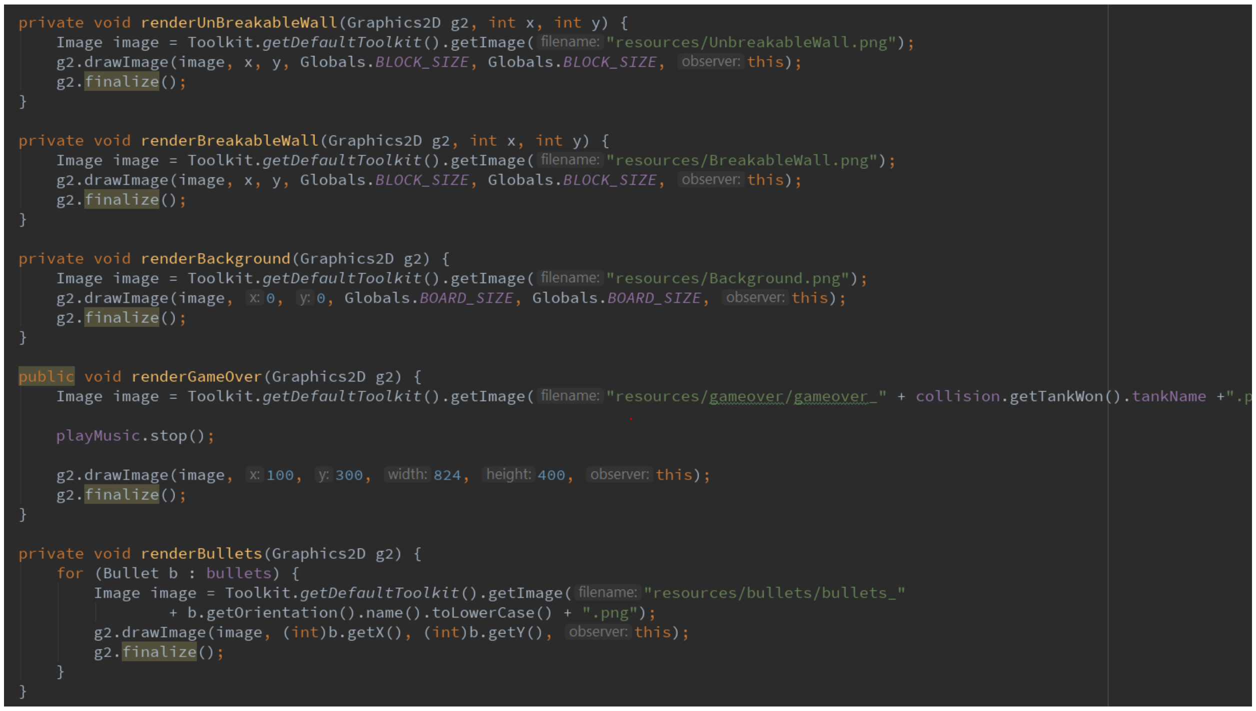
```
public void handleMovement(Graphics g){
    int newX, newY;
    int oldX, oldY ;

    count++;
    if(count == frame){
        count = 0;
    } else{
        return;
    }

    if (this.tank2movingup) {
        newX = tank2.x;
        newY = tank2.y - Globals.BLOCK_SIZE;
        oldY = tank2.y;
        if (collision.validateCollision(newX, newY, tank1)) {
            tank2.y = oldY;
        }else {
            tank2.y = newY;
        }
        tank2.orientation = TankOrientation.TOP;
    }

    if (this.tank2movingdown){
        newX = tank2.x;
        newY = tank2.y + Globals.BLOCK_SIZE ;
        oldY = tank2.y;
        if (collision.validateCollision(newX, newY, tank1)) {
           tank2.y = oldY;
        }else {
            tank2.y = newY;
        }
        tank2.orientation = TankOrientation.DOWN;
    }
```

moveBullets(): If the iterator of bullets has next bullet in it, then this function validates if there is any collision between bullets to tank or wall, if yes then it displays an explosion animation on the screen and plays the explosion sound, and removes that bullet from the iterator. If no collision, then bullet keeps moving.



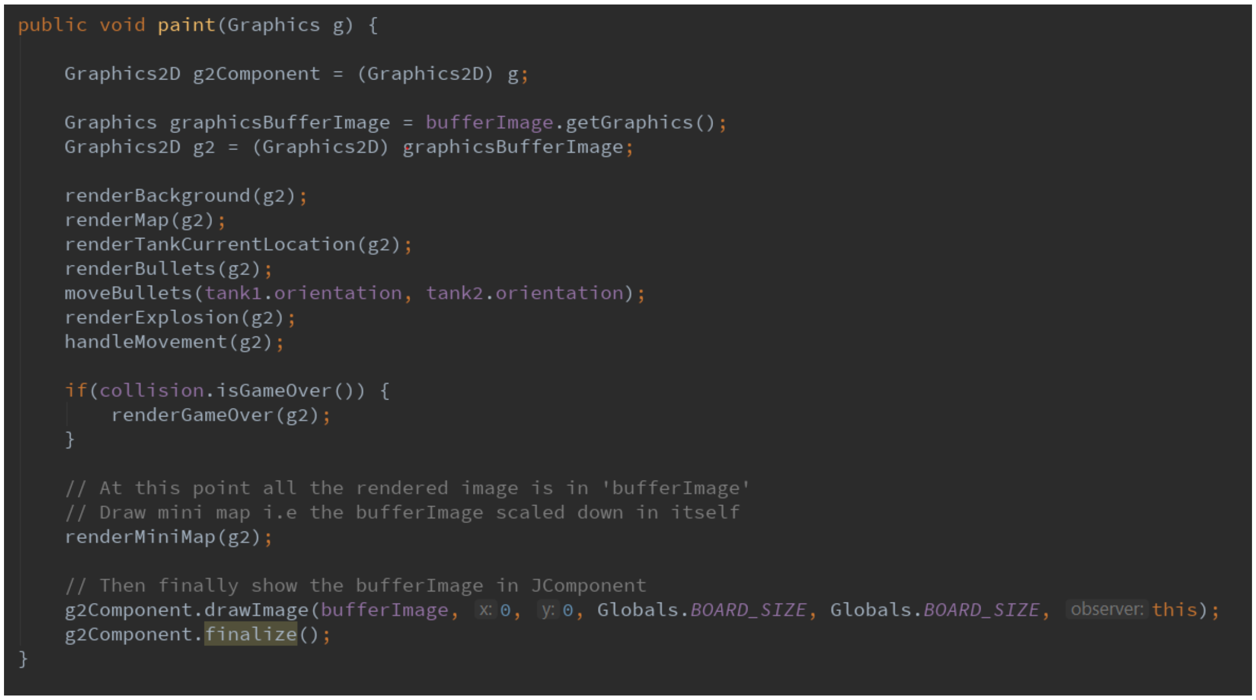
```java
public void moveBullets(TankOrientation tankOrientation1, TankOrientation tankOrientation2) {
    assert tankOrientation1 != null : "tank1 orientation cannot be null";
    assert tankOrientation2 != null : "tank2 orientation cannot be null";
    Iterator<Bullet> iter = bullets.iterator();

    while (iter.hasNext()) {
        Bullet bullet = iter.next();
        if (collision.validateBullettoWallCollision(bullet) ||
                collision.validateBullettoTankCollision(bullet, tank1, tank2)) {
            Explosion explosion = new Explosion(bullet.getX(),bullet.getY());
            explosions.add(explosion);
            explosionSound.play();
            iter.remove();
        } else {
            bullet.moveBullet();
        }
    }
}
```

Some render functions as below:



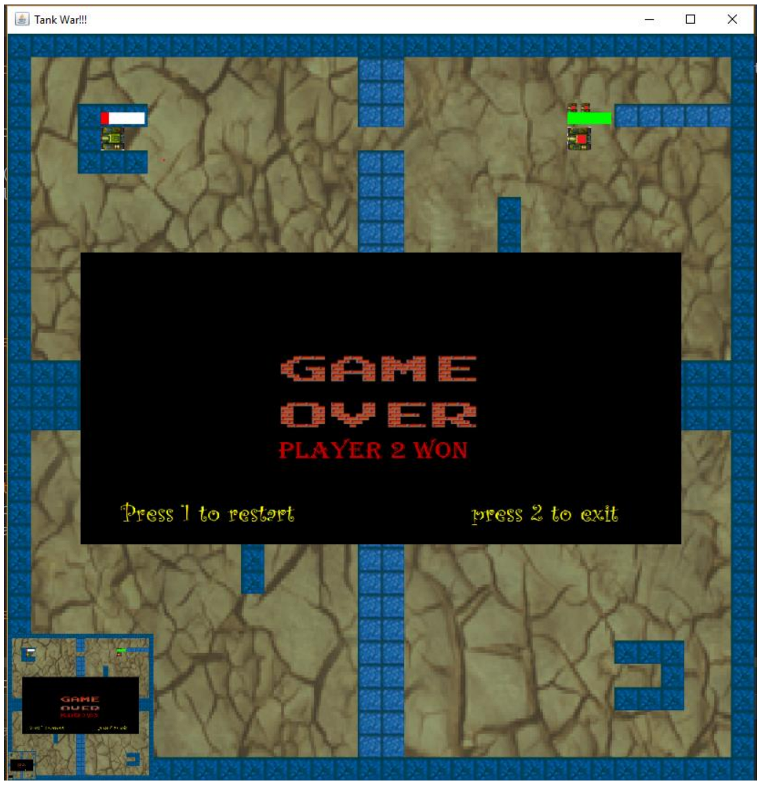
```java
private void renderUnBreakableWall(Graphics2D g2, int x, int y) {
    Image image = Toolkit.getDefaultToolkit().getImage( filename: "resources/UnbreakableWall.png");
    g2.drawImage(image, x, y, Globals.BLOCK_SIZE, Globals.BLOCK_SIZE,  observer: this);
    g2.finalize();
}

private void renderBreakableWall(Graphics2D g2, int x, int y) {
    Image image = Toolkit.getDefaultToolkit().getImage( filename: "resources/BreakableWall.png");
    g2.drawImage(image, x, y, Globals.BLOCK_SIZE, Globals.BLOCK_SIZE,  observer: this);
    g2.finalize();
}

private void renderBackground(Graphics2D g2) {
    Image image = Toolkit.getDefaultToolkit().getImage( filename: "resources/Background.png");
    g2.drawImage(image,  x: 0,  y: 0, Globals.BOARD_SIZE, Globals.BOARD_SIZE,  observer: this);
    g2.finalize();
}

public void renderGameOver(Graphics2D g2) {
    Image image = Toolkit.getDefaultToolkit().getImage( filename: "resources/gameover/gameover_" + collision.getTankWon().tankName +".p

    playMusic.stop();

    g2.drawImage(image,  x: 100,  y: 300,  width: 824,  height: 400,  observer: this);
    g2.finalize();
}

private void renderBullets(Graphics2D g2) {
    for (Bullet b : bullets) {
        Image image = Toolkit.getDefaultToolkit().getImage( filename: "resources/bullets/bullets_"
                + b.getOrientation().name().toLowerCase() + ".png");
        g2.drawImage(image, (int)b.getX(), (int)b.getY(),  observer: this);
        g2.finalize();
    }
}
```

paint(): calls all the render functions and paints all the objects on the JFrame window.

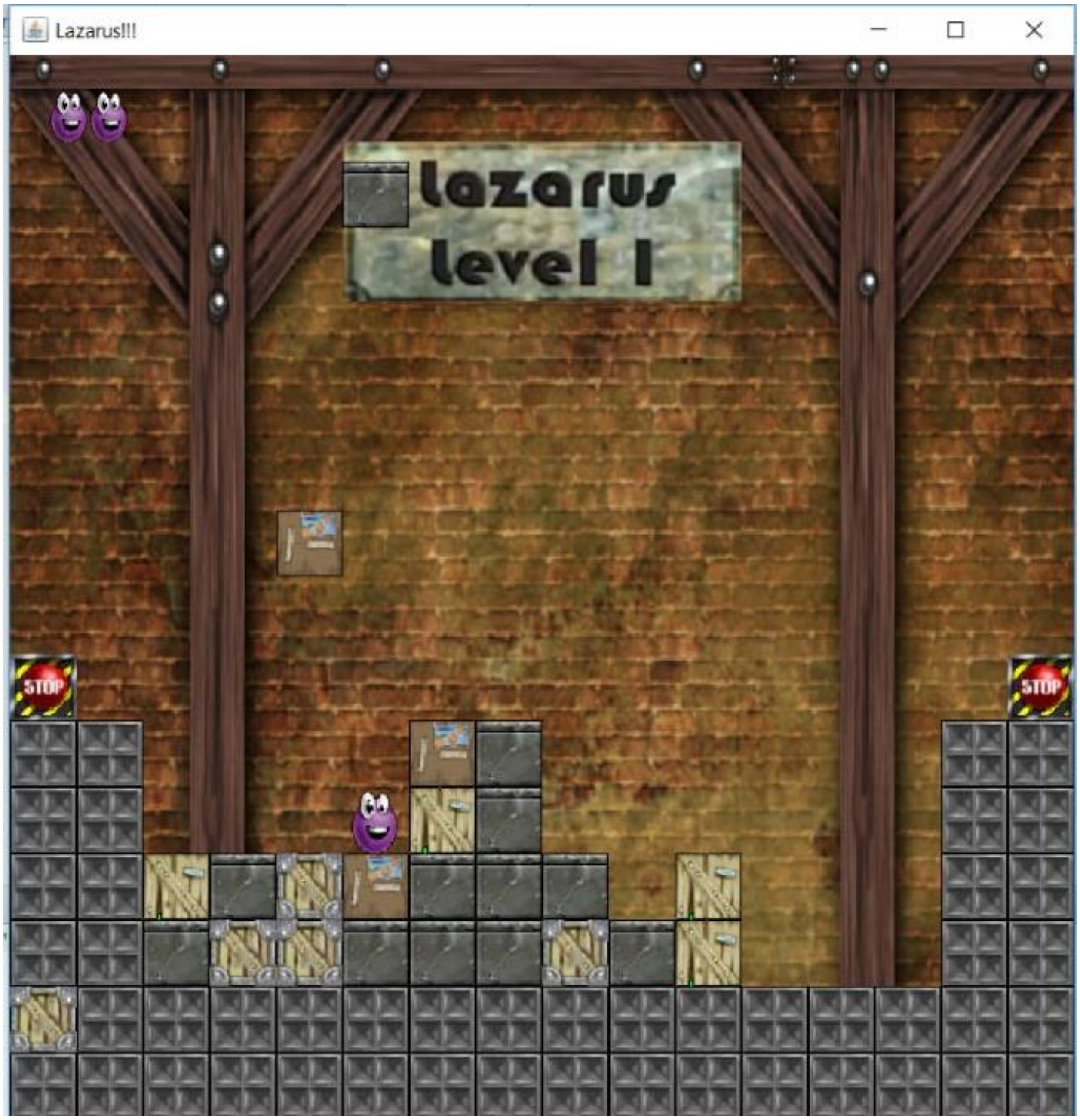


```
public void paint(Graphics g) {

    Graphics2D g2Component = (Graphics2D) g;

    Graphics graphicsBufferImage = bufferImage.getGraphics();
    Graphics2D g2 = (Graphics2D) graphicsBufferImage;

    renderBackground(g2);
    renderMap(g2);
    renderTankCurrentLocation(g2);
    renderBullets(g2);
    moveBullets(tank1.orientation, tank2.orientation);
    renderExplosion(g2);
    handleMovement(g2);

    if(collision.isGameOver()) {
        renderGameOver(g2);
    }

    // At this point all the rendered image is in 'bufferImage'
    // Draw mini map i.e the bufferImage scaled down in itself
    renderMiniMap(g2);

    // Then finally show the bufferImage in JComponent
    g2Component.drawImage(bufferImage, x: 0, y: 0, Globals.BOARD_SIZE, Globals.BOARD_SIZE, observer: this);
    g2Component.finalize();
}
```

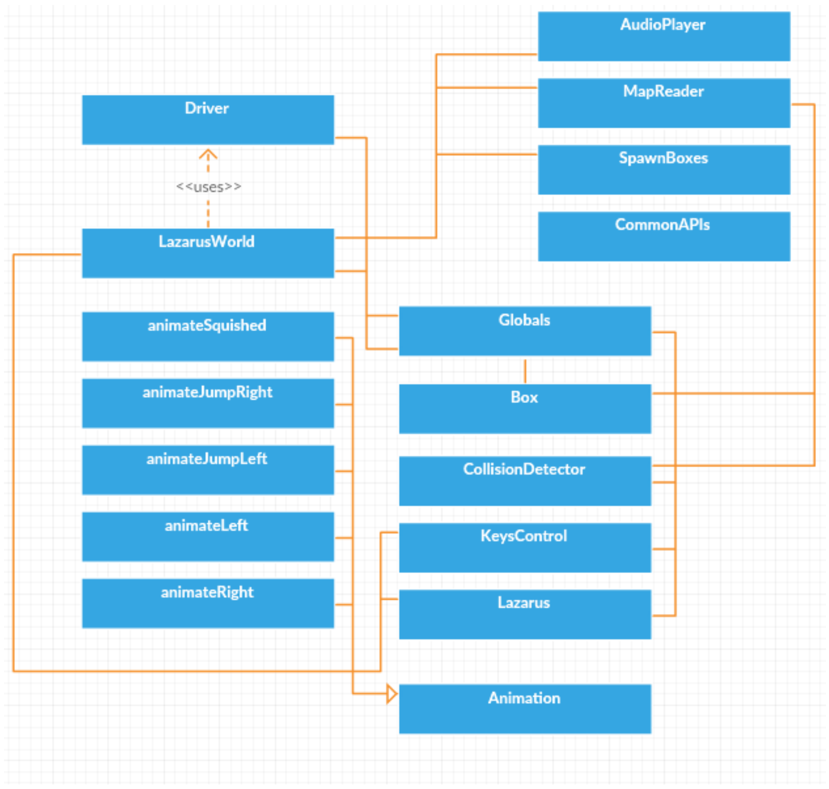
Game over screen:



Lazarus Game:



Lazarus Class Diagram:



Class Overview:

commons package:

AudioPlayer Class
+Audioplayer +Audioplayer(TankWorld wingman, String filename) +play(): void +loop():void +stop():void

We have reused this whole class from the Tank game. This class loads, plays and loops the sound file within the game.

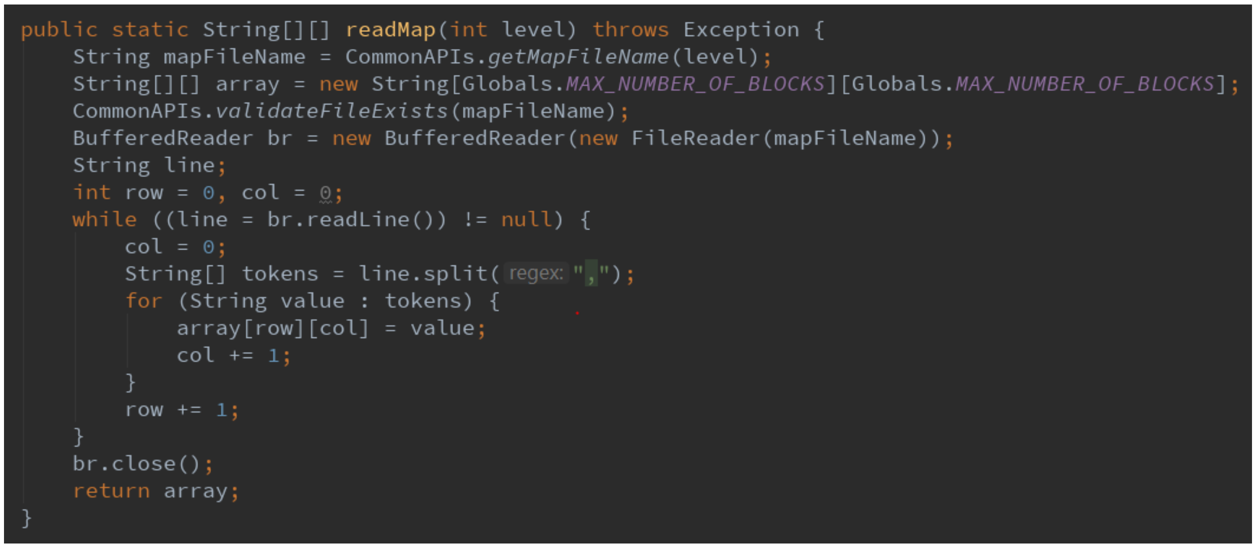
CommonAPIs Class
+validateFileExits(String filename):void +loadImages(String path): BufferedImage +getMapFimeName(int level): String

Similar to the CommonAPIs class from the Tank game that contains some common functions used all over the project.

Globals Class:
Similar to the Tank game's Globals class, this class has all the global variables in it.

MapReader Class
+readMap(int level):static String[][]

Similar to the Tank's MapReader class, this class takes the input as .csv file and locates the wall and Lazarus for the level of the game.

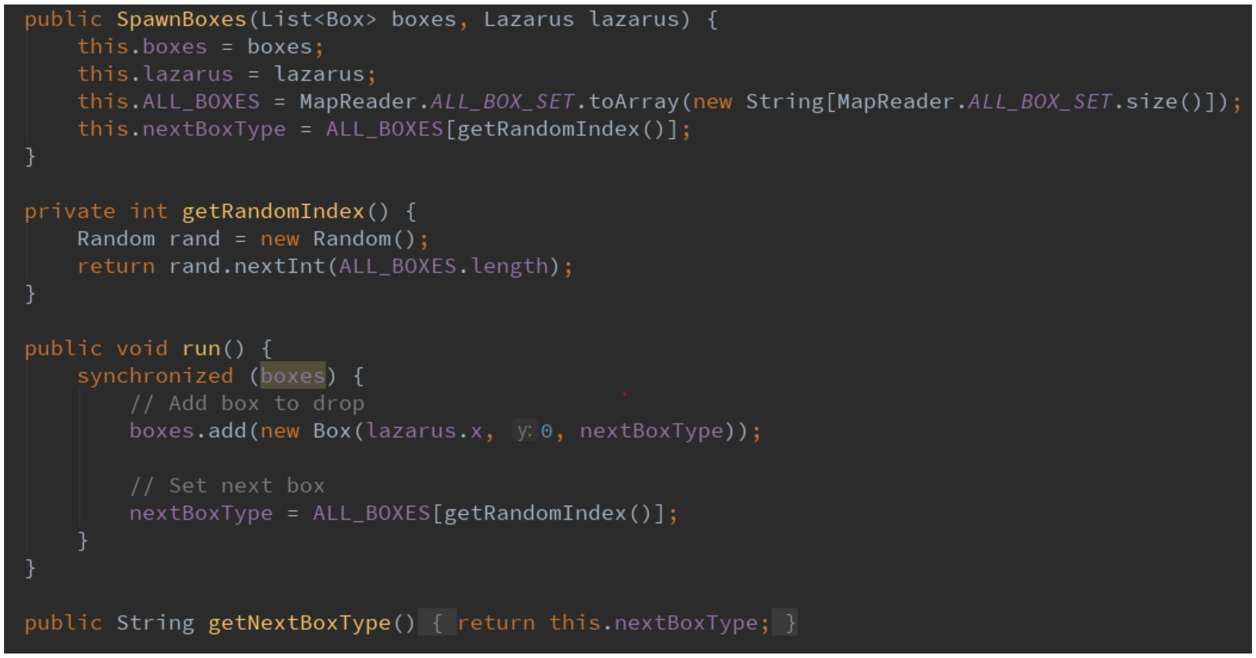


```java
public static String[][] readMap(int level) throws Exception {
    String mapFileName = CommonAPIs.getMapFileName(level);
    String[][] array = new String[Globals.MAX_NUMBER_OF_BLOCKS][Globals.MAX_NUMBER_OF_BLOCKS];
    CommonAPIs.validateFileExists(mapFileName);
    BufferedReader br = new BufferedReader(new FileReader(mapFileName));
    String line;
    int row = 0, col = 0;
    while ((line = br.readLine()) != null) {
        col = 0;
        String[] tokens = line.split( regex: ",");
        for (String value : tokens) {
            array[row][col] = value;
            col += 1;
        }
        row += 1;
    }
    br.close();
    return array;
}
```

SpawnBoxes Class
-getRandomIndex():int +run():void

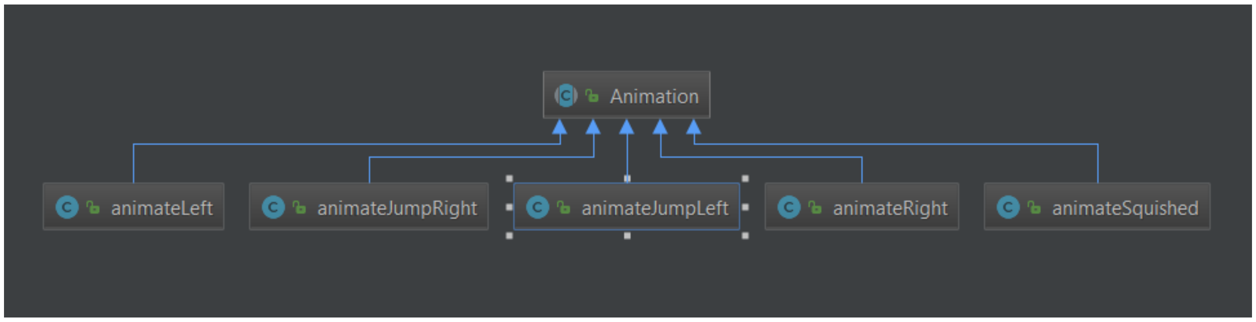
getRandomIndex method: generates the random integer and returns the value. This method is called in the run method of the SpawnBoxes class to generate random boxes.

run(): generates box after 2000 milliseconds and adds this new box into an arraylist. The boxes are generated by recognizing the Lazarus's position.



```
public SpawnBoxes(List<Box> boxes, Lazarus lazarus) {
    this.boxes = boxes;
    this.lazarus = lazarus;
    this.ALL_BOXES = MapReader.ALL_BOX_SET.toArray(new String[MapReader.ALL_BOX_SET.size()]);
    this.nextBoxType = ALL_BOXES[getRandomIndex()];
}

private int getRandomIndex() {
    Random rand = new Random();
    return rand.nextInt(ALL_BOXES.length);
}

public void run() {
    synchronized (boxes) {
        // Add box to drop
        boxes.add(new Box(lazarus.x,  y: 0, nextBoxType));

        // Set next box
        nextBoxType = ALL_BOXES[getRandomIndex()];
    }
}

public String getNextBoxType() { return this.nextBoxType; }
```

component package:



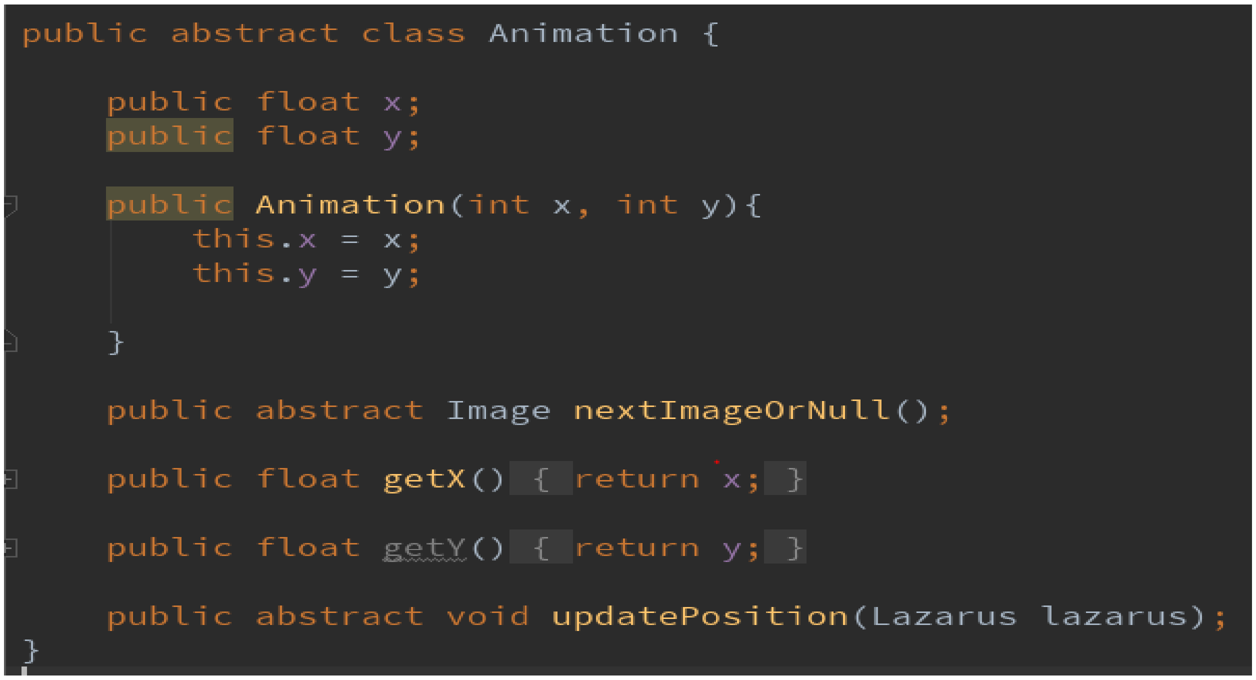
Animation class has been derived to 5 animated classes including animateLeft, animateRight, animateJumpLeft, animateJumpRight and animateSquished. Each subclass implements the abstract method from the Animation class.

Animation Class
+nextImageOrNull(): Image +getX(): float +getY(): float +updatePosition(Lazarus lazarus): void

The classes that inherit the Animation class have a constructor that breaks down the gif image into the several frames and assigned to an array.

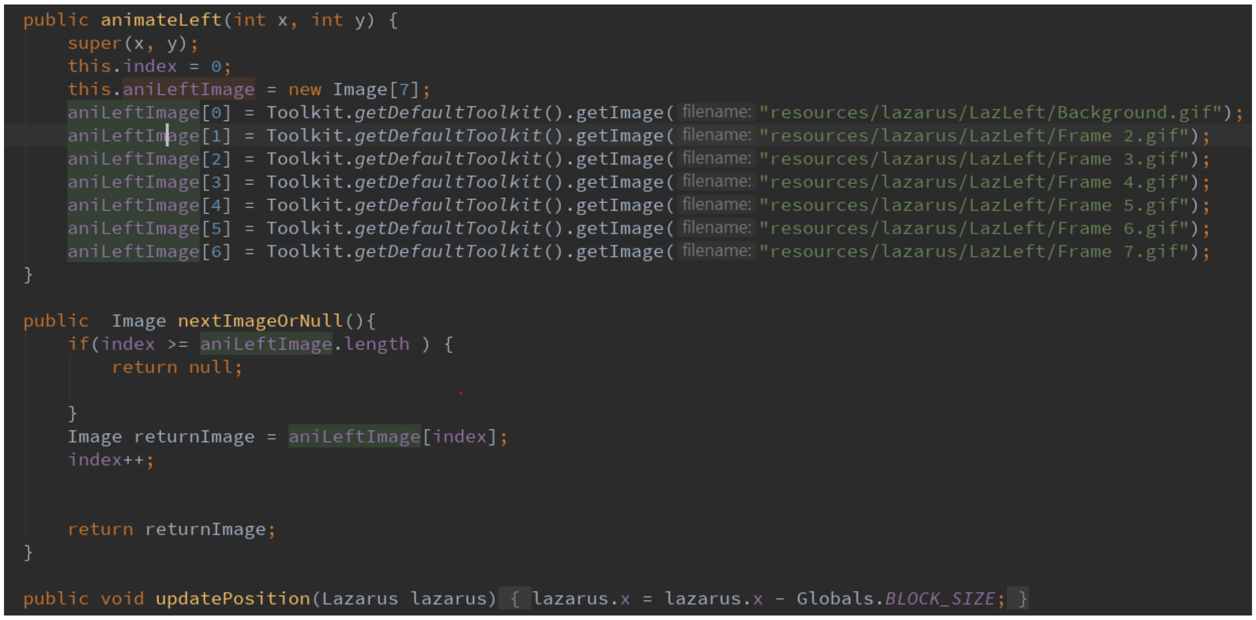
nextImageOrNull(): checks if there's next image.

updatePosition(lazarus): depending on which key is pressed (left or right), it will update the image of lazarus if the movement is valid.



```java
public abstract class Animation {

    public float x;
    public float y;

    public Animation(int x, int y){
        this.x = x;
        this.y = y;

    }

    public abstract Image nextImageOrNull();

    public float getX() { return x; }

    public float getY() { return y; }

    public abstract void updatePosition(Lazarus lazarus);
}
```

One of the child of the Animation class:



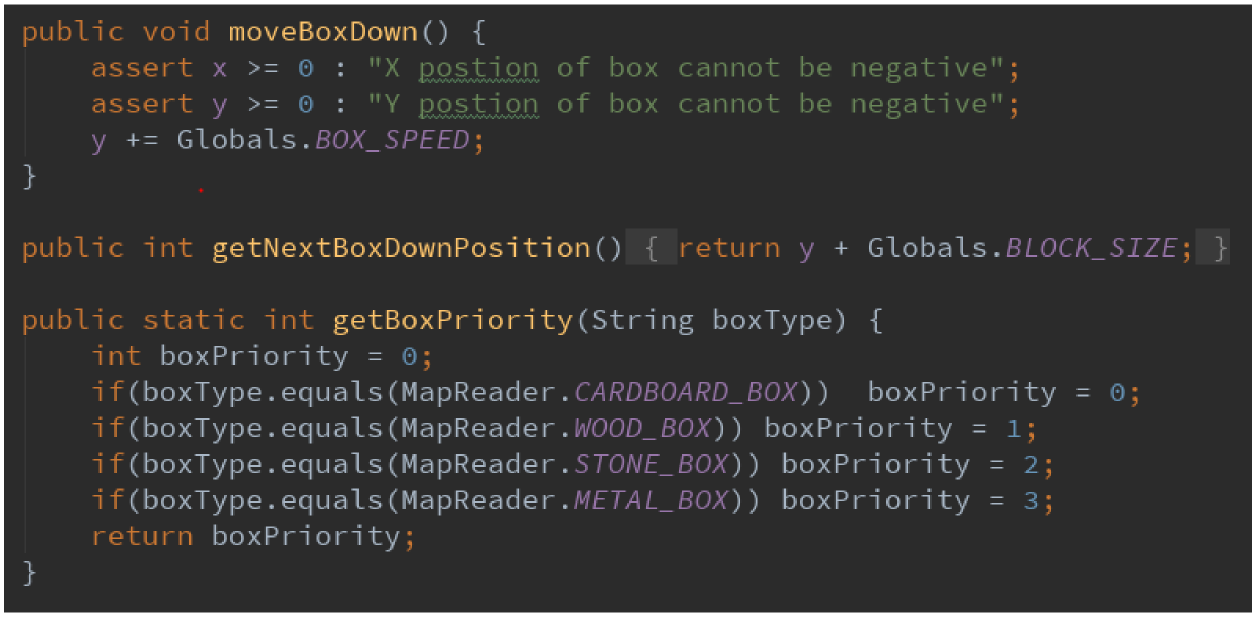
```
public animateLeft(int x, int y) {
    super(x, y);
    this.index = 0;
    this.aniLeftImage = new Image[7];
    aniLeftImage[0] = Toolkit.getDefaultToolkit().getImage( filename: "resources/lazarus/LazLeft/Background.gif");
    aniLeftImage[1] = Toolkit.getDefaultToolkit().getImage( filename: "resources/lazarus/LazLeft/Frame 2.gif");
    aniLeftImage[2] = Toolkit.getDefaultToolkit().getImage( filename: "resources/lazarus/LazLeft/Frame 3.gif");
    aniLeftImage[3] = Toolkit.getDefaultToolkit().getImage( filename: "resources/lazarus/LazLeft/Frame 4.gif");
    aniLeftImage[4] = Toolkit.getDefaultToolkit().getImage( filename: "resources/lazarus/LazLeft/Frame 5.gif");
    aniLeftImage[5] = Toolkit.getDefaultToolkit().getImage( filename: "resources/lazarus/LazLeft/Frame 6.gif");
    aniLeftImage[6] = Toolkit.getDefaultToolkit().getImage( filename: "resources/lazarus/LazLeft/Frame 7.gif");
}

public  Image nextImageOrNull(){
    if(index >= aniLeftImage.length ) {
        return null;

    }
    Image returnImage = aniLeftImage[index];
    index++;


    return returnImage;
}

public void updatePosition(Lazarus lazarus) { lazarus.x = lazarus.x - Globals.BLOCK_SIZE; }
```

Box Class
+moveBoxDown(): void +getNextBoxDownPosition(): int +getBoxPriority(String boxType): int +getX(): int +getY(): int +getBoxType(): String

moveBoxDown(): gives motion to the boxes and lets boxes fall down by the BOX_SPEED

getNextBoxDownPosition(): returns next box’s position

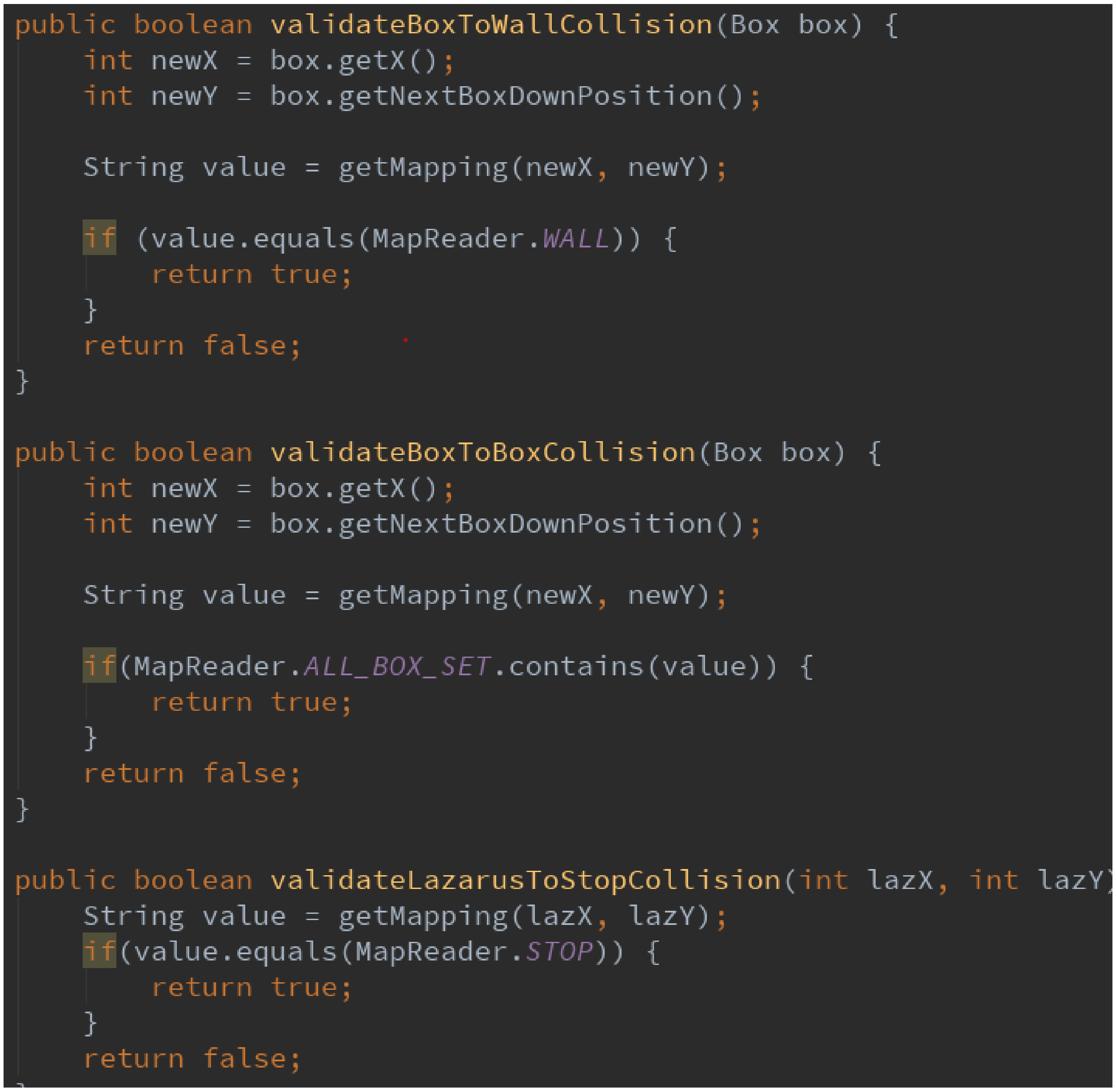
getBoxPriority(): returns the priority of the box. This priority is used to check if the box is heavy or light.



```
public void moveBoxDown() {
    assert x >= 0 : "X postion of box cannot be negative";
    assert y >= 0 : "Y postion of box cannot be negative";
    y += Globals.BOX_SPEED;
}

public int getNextBoxDownPosition() { return y + Globals.BLOCK_SIZE; }

public static int getBoxPriority(String boxType) {
    int boxPriority = 0;
    if(boxType.equals(MapReader.CARDBOARD_BOX))  boxPriority = 0;
    if(boxType.equals(MapReader.WOOD_BOX)) boxPriority = 1;
    if(boxType.equals(MapReader.STONE_BOX)) boxPriority = 2;
    if(boxType.equals(MapReader.METAL_BOX)) boxPriority = 3;
    return boxPriority;
}
```

CollisionDetector Class
+validateLazarusCollision(int newX, int newY): Boolean +validateLazarustoBoxesCollision(int newX, int newY): Boolean +getMapping(int newX, int newY): String +validateBoxToWallCollision(Box box): Boolean +validateBoxToBoxCollision(Box box): Boolean +validateLazarustoSTPCollision(int lazX, int lazY): Boolean
-validateLazarusToBoundaryCollision(int newX, int newY): Boolean -validateLazarusToWallCollision(int newX, int newY): Boolean

CollisionDector class has the same concept as the CollisionDector of the Tank game. This class checks the collision between lazarus to wall, boundary and boxes, and returns true if there is any collision. It also checks the collision between the Lazarus and Stop block. If the Lazarus collides with the Stop block, then the level ends, player wins the level, and it asks for user input to move to the next level.

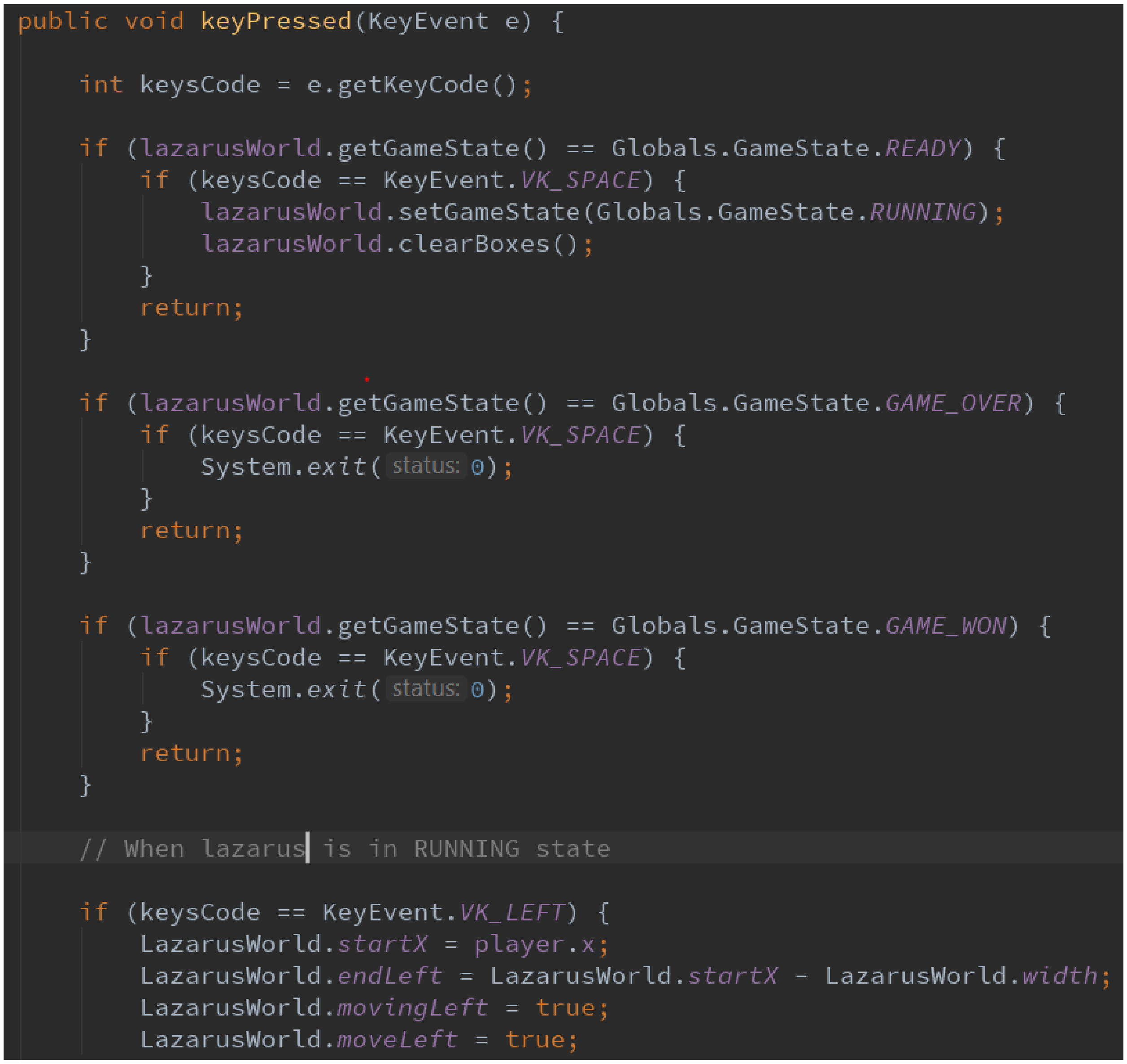


```java
public boolean validateBoxToWallCollision(Box box) {
    int newX = box.getX();
    int newY = box.getNextBoxDownPosition();

    String value = getMapping(newX, newY);

    if (value.equals(MapReader.WALL)) {
        return true;
    }
    return false;
}

public boolean validateBoxToBoxCollision(Box box) {
    int newX = box.getX();
    int newY = box.getNextBoxDownPosition();

    String value = getMapping(newX, newY);

    if(MapReader.ALL_BOX_SET.contains(value)) {
        return true;
    }
    return false;
}

public boolean validateLazarusToStopCollision(int lazX, int lazY)
    String value = getMapping(lazX, lazY);
    if(value.equals(MapReader.STOP)) {
        return true;
    }
    return false;
```

KeysControl Class
+keyPressed(KeyEvent e): void +keyReleased(KeyEvent e): void

Similar to the KeysControl class of the Tank game. It assigns keys to the Lazarus's movements.

Lazarus left/jump left	Left arrow
Lazarus right/jump right	Right arrow
Space bar	Start game, Next Level, Game Exit

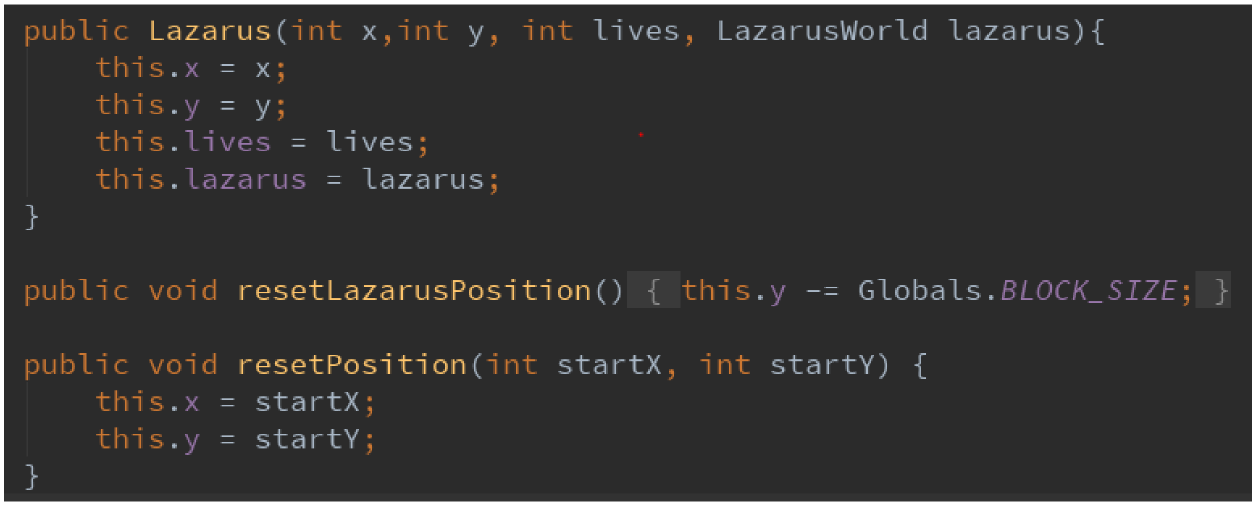


```
public void keyPressed(KeyEvent e) {

    int keysCode = e.getKeyCode();

    if (lazarusWorld.getGameState() == Globals.GameState.READY) {
        if (keysCode == KeyEvent.VK_SPACE) {
            lazarusWorld.setGameState(Globals.GameState.RUNNING);
            lazarusWorld.clearBoxes();
        }
        return;
    }

    if (lazarusWorld.getGameState() == Globals.GameState.GAME_OVER) {
        if (keysCode == KeyEvent.VK_SPACE) {
            System.exit( status: 0);
        }
        return;
    }

    if (lazarusWorld.getGameState() == Globals.GameState.GAME_WON) {
        if (keysCode == KeyEvent.VK_SPACE) {
            System.exit( status: 0);
        }
        return;
    }

    // When lazarus is in RUNNING state

    if (keysCode == KeyEvent.VK_LEFT) {
        LazarusWorld.startX = player.x;
        LazarusWorld.endLeft = LazarusWorld.startX - LazarusWorld.width;
        LazarusWorld.movingLeft = true;
        LazarusWorld.moveLeft = true;
```

Lazarus Class
+Lazarus(int x, int y, int lives, LazarusWorld lazarus) +resetLazarusPosition(): void +resetPosition(int startX, int startY): void

This class contains every properties of the lazarus including lazarus x, y positions and lives.

resetLazarusPosition(): resets the Lazarus's position above the box when Lazarus dies.

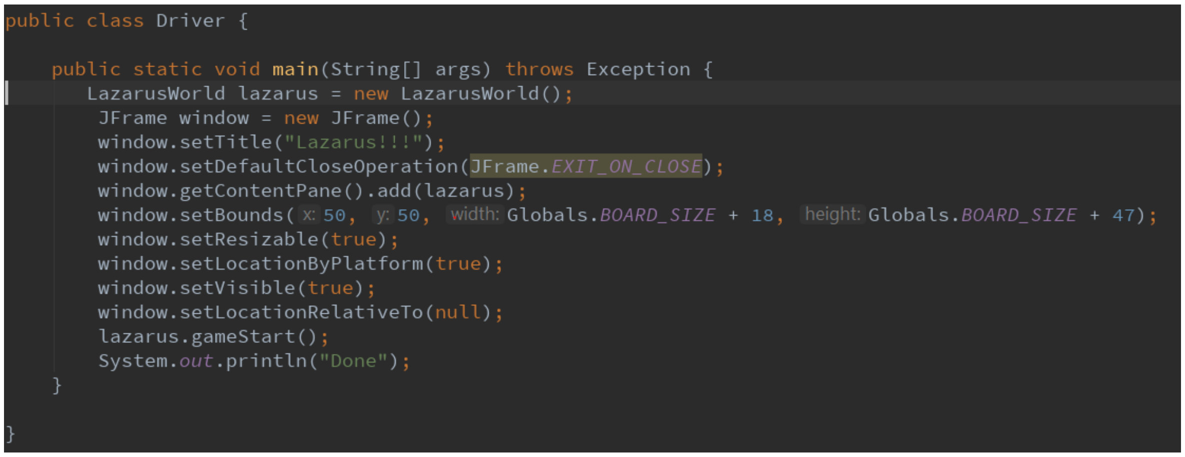


```
public Lazarus(int x,int y, int lives, LazarusWorld lazarus){
    this.x = x;
    this.y = y;
    this.lives = lives;
    this.lazarus = lazarus;
}

public void resetLazarusPosition() { this.y -= Globals.BLOCK_SIZE; }

public void resetPosition(int startX, int startY) {
    this.x = startX;
    this.y = startY;
}
```

core package:

Driver Class:

Similar to the Driver class of the Tank game. It uses Java Swing for creating JFrame window.



```
public class Driver {

    public static void main(String[] args) throws Exception {
        LazarusWorld lazarus = new LazarusWorld();
        JFrame window = new JFrame();
        window.setTitle("Lazarus!!!");
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        window.getContentPane().add(lazarus);
        window.setBounds( x: 50,  y: 50,  width: Globals.BOARD_SIZE + 18,  height: Globals.BOARD_SIZE + 47);
        window.setResizable(true);
        window.setLocationByPlatform(true);
        window.setVisible(true);
        window.setLocationRelativeTo(null);
        lazarus.gameStart();
        System.out.println("Done");
    }

}
```

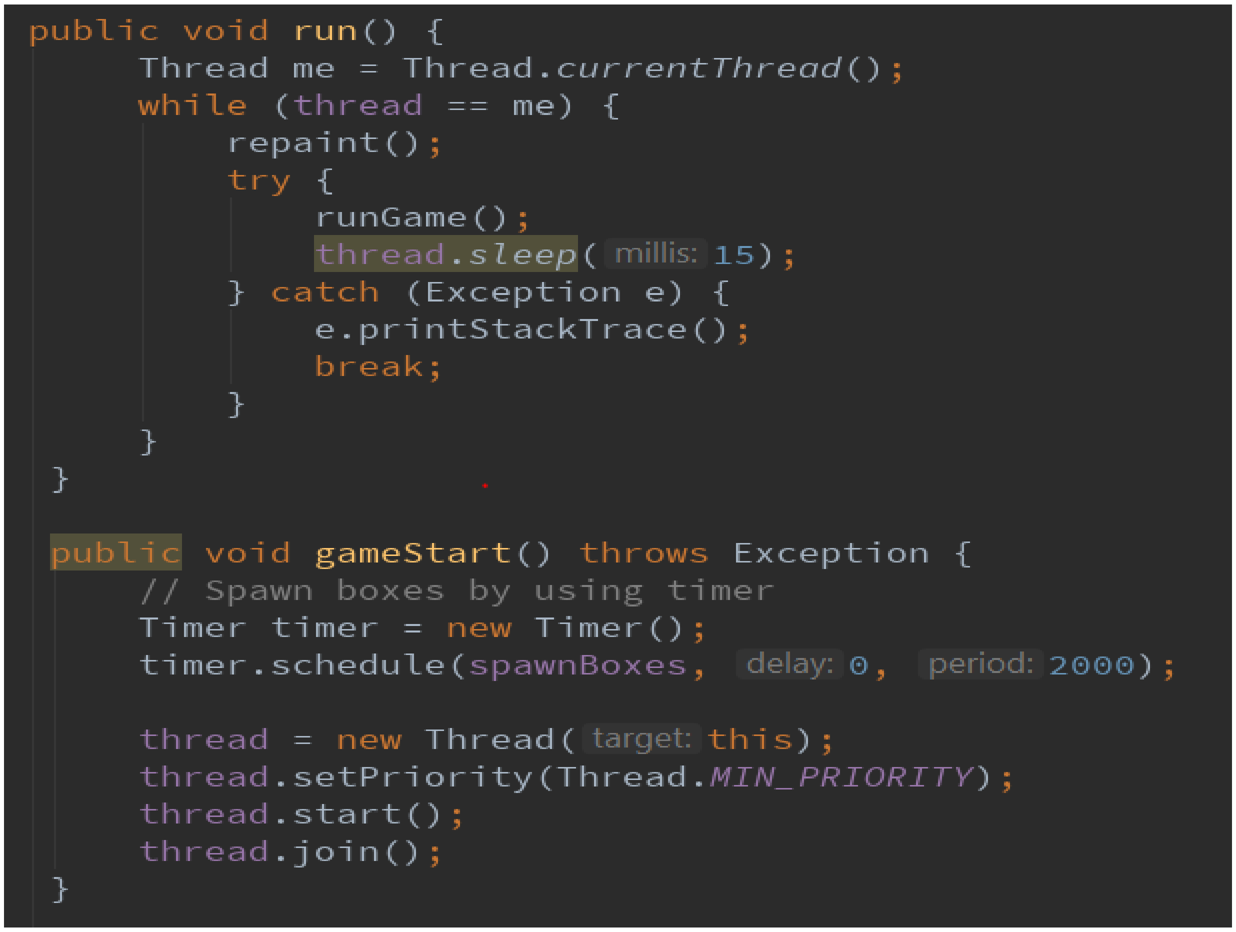
LazarusWorld Class
+LazarusWorld() +gameReset(int nextLevel): void +moveToNextLevel(): void +clearBoxes(): void +setGameState(Globals.GameState state): void +getGameState(): Globals.GameState +paint(Graphic g): void +handleMovement(): void +renderBackground(Graphics2D g2): void +renderLevel(Graphics2D g2): void +findStartPosition(): void +run(): void +gameStart(): void
-runGame(): void -renderLives(Graphics2D g2): void -renderGameWonScreen(Graphics2D g2): void -renderGameOverScreen(Graphics2D g2): void -renderReadyScreen(Graphics2D g2): void -renderNextBox(Graphics2D g2): void -renderBoxes(Graphics2D g2): void -renderBox(Graphics2D g2, String boxType, int newX, int newY): void -lazarusDie(): void -moveBoxes(): void -stopBoxToBoxOnCollision(Box currentBox, Iterator<Box> itr): void -renderMap(Graphics2D g2): void -renderWall(Graphics2D g2): void -renderButton(Graphics2D g2): void -drawLazarus(Graphics2D g2, int x, int y): void

Similar to the TankWorld of the Tank game. This class is the main class of the Lazarus game that implements runnable interface. It binds all the objects of the game.

In the constructor, all the variables, map, level, sound and all other classes are initialized. An object of LazarusWorld is passed to the main method of Driver class, and start() method of LazarusWorld is called there.

start() method: a Timer is scheduled for spawning the boxes by an interval of 2000 milliseconds. Also, a new/main thread is created in this method, and it starts to run the game and to paint it on the canvas.

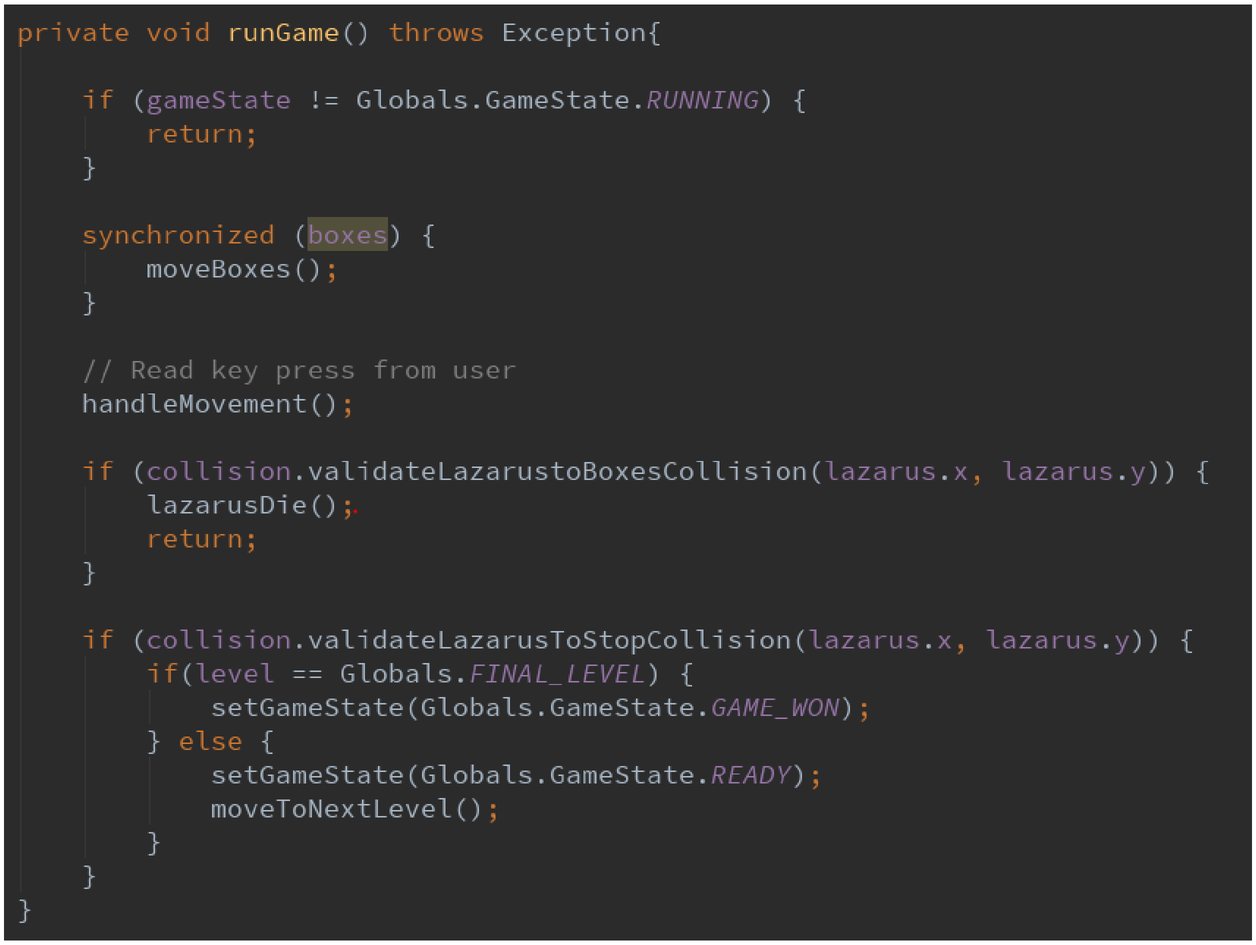
run() method: repaints all the game continuously on the JFrame window. In this method, runGame() method is called which actually runs the game by moving Lazarus, boxes etc.



```
public void run() {
    Thread me = Thread.currentThread();
    while (thread == me) {
        repaint();
        try {
            runGame();
            thread.sleep( millis: 15);
        } catch (Exception e) {
            e.printStackTrace();
            break;
        }
    }
}

public void gameStart() throws Exception {
    // Spawn boxes by using timer
    Timer timer = new Timer();
    timer.schedule(spawnBoxes, delay: 0, period: 2000);

    thread = new Thread( target: this);
    thread.setPriority(Thread.MIN_PRIORITY);
    thread.start();
    thread.join();
}
```

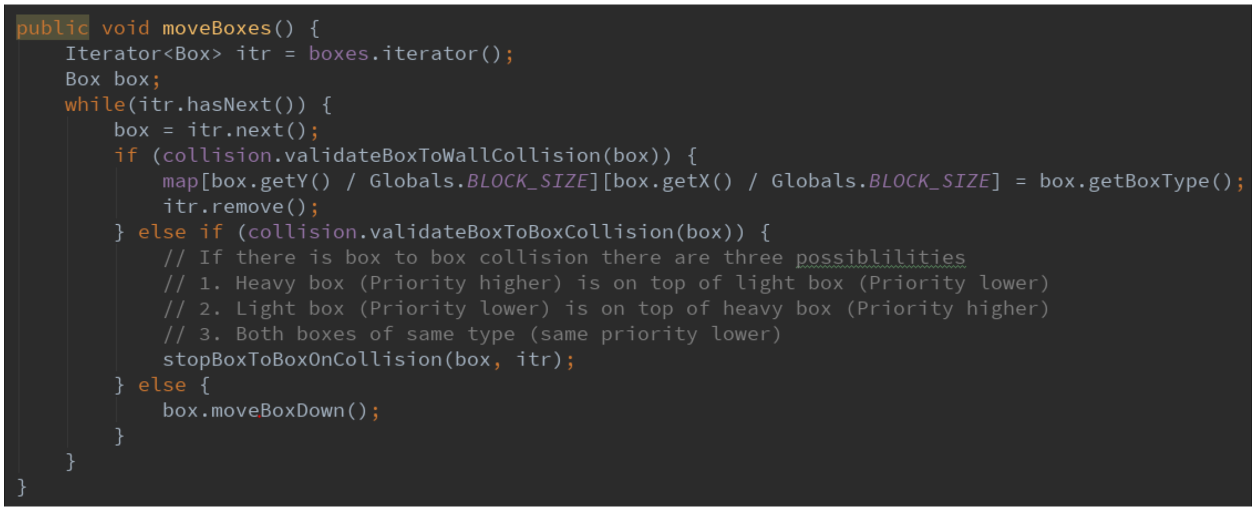
runGame() method: if the game is in the running state, then this method calls:

- moveBoxes() method: moves boxes in the downward direction.
- handleMovement(): handles the Lazarus movement
- lazarusDie(): validates if the boxes are falling on Lazarus, if yes, then lazarus dies.
- Validates Lazarus to STOP block collision, if true, then checks if the player has won the final level or the 1st level. If final level, then sets the Game state as GAME_WON and displays a message accordingly on the window. If player has won the 1st level, then sets the Game state as READY and displays the message accordingly on the window and moves to the next level.



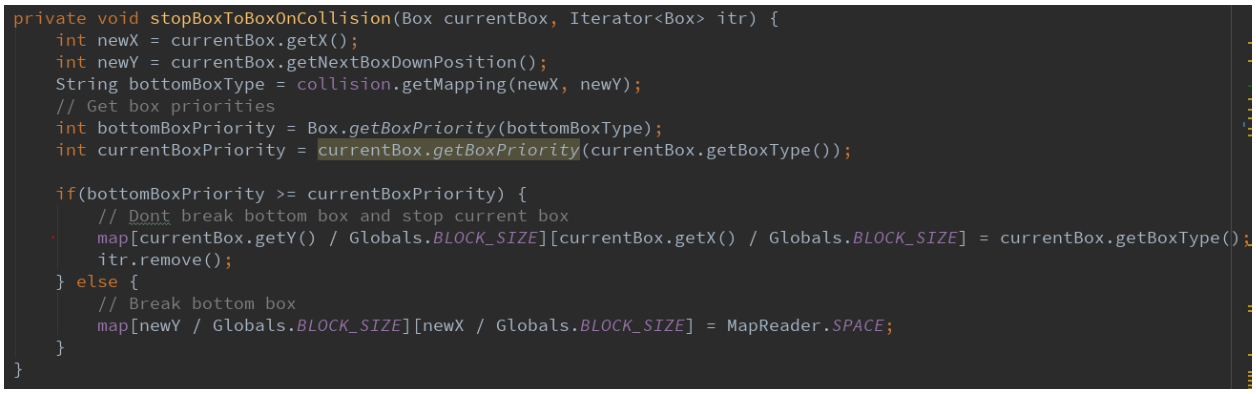
```java
private void runGame() throws Exception{

    if (gameState != Globals.GameState.RUNNING) {
        return;
    }

    synchronized (boxes) {
        moveBoxes();
    }

    // Read key press from user
    handleMovement();

    if (collision.validateLazarustoBoxesCollision(lazarus.x, lazarus.y)) {
        lazarusDie();
        return;
    }

    if (collision.validateLazarusToStopCollision(lazarus.x, lazarus.y)) {
        if(level == Globals.FINAL_LEVEL) {
            setGameState(Globals.GameState.GAME_WON);
        } else {
            setGameState(Globals.GameState.READY);
            moveToNextLevel();
        }
    }
}
```

moveBoxes() method: iterates over an arraylist. If there is a collision between box to wall, then removes the box from the iterator. If there is a collision between box to box, then calls stopBoxToBoxOnCollision() method. Otherwise, lets box fall down.



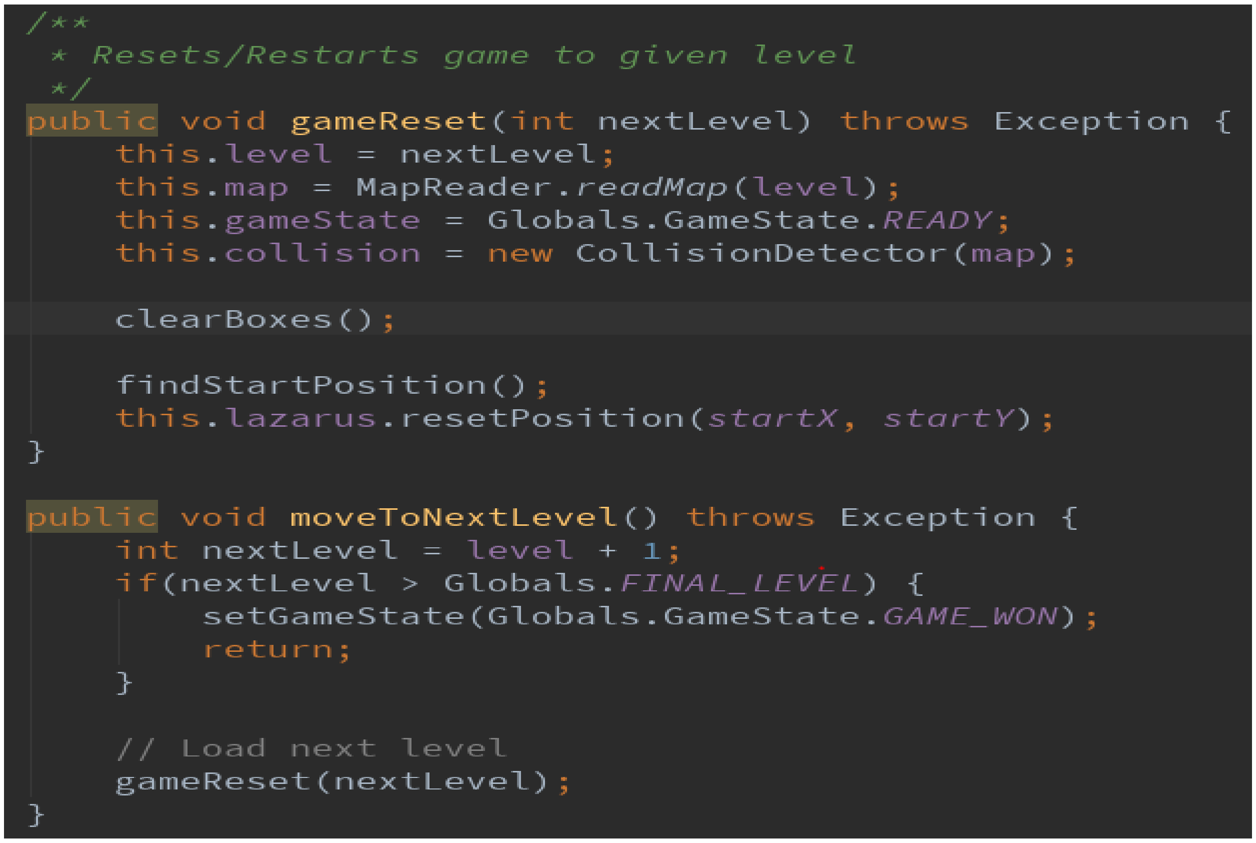
```java
public void moveBoxes() {
    Iterator<Box> itr = boxes.iterator();
    Box box;
    while(itr.hasNext()) {
        box = itr.next();
        if (collision.validateBoxToWallCollision(box)) {
            map[box.getY() / Globals.BLOCK_SIZE][box.getX() / Globals.BLOCK_SIZE] = box.getBoxType();
            itr.remove();
        } else if (collision.validateBoxToBoxCollision(box)) {
            // If there is box to box collision there are three possiblilities
            // 1. Heavy box (Priority higher) is on top of light box (Priority lower)
            // 2. Light box (Priority lower) is on top of heavy box (Priority higher)
            // 3. Both boxes of same type (same priority lower)
            stopBoxToBoxOnCollision(box, itr);
        } else {
            box.moveBoxDown();
        }
    }
}
```

stopBoxToBoxOnCollision(): gets the position of current box and next box which is below the current box. Checks the priority of both the boxes. If the bottom box priority is greater than or equal to the current box, then current box sits on the top of the bottom box, and the iterator removes the current box. If the current box's priority is greater, then disappear the bottom box.

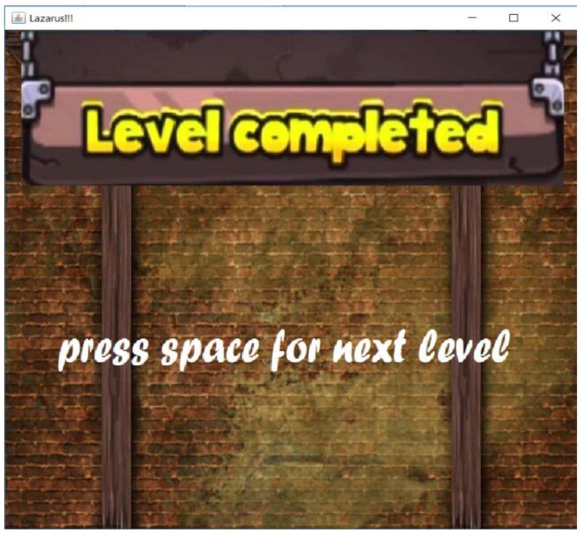


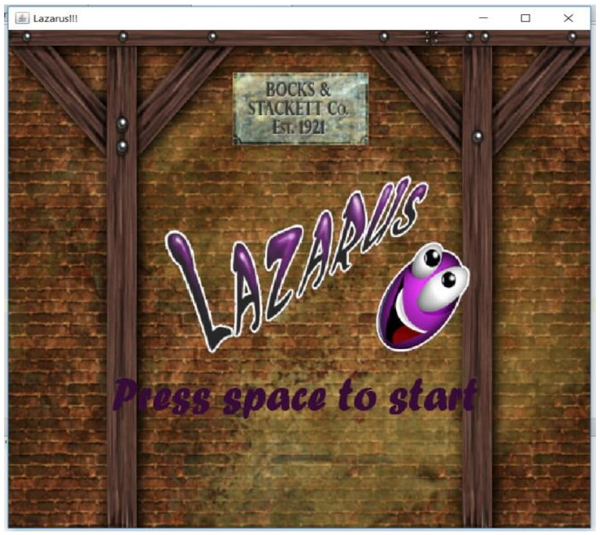
```java
private void stopBoxToBoxOnCollision(Box currentBox, Iterator<Box> itr) {
    int newX = currentBox.getX();
    int newY = currentBox.getNextBoxDownPosition();
    String bottomBoxType = collision.getMapping(newX, newY);
    // Get box priorities
    int bottomBoxPriority = Box.getBoxPriority(bottomBoxType);
    int currentBoxPriority = currentBox.getBoxPriority(currentBox.getBoxType());

    if(bottomBoxPriority >= currentBoxPriority) {
        // Dont break bottom box and stop current box
        map[currentBox.getY() / Globals.BLOCK_SIZE][currentBox.getX() / Globals.BLOCK_SIZE] = currentBox.getBoxType();
        itr.remove();
    } else {
        // Break bottom box
        map[newY / Globals.BLOCK_SIZE][newX / Globals.BLOCK_SIZE] = MapReader.SPACE;
    }
}
```

gameReset() method: resets or restarts the game to the given level.
moveToNextLevel() method: moves to the next level.

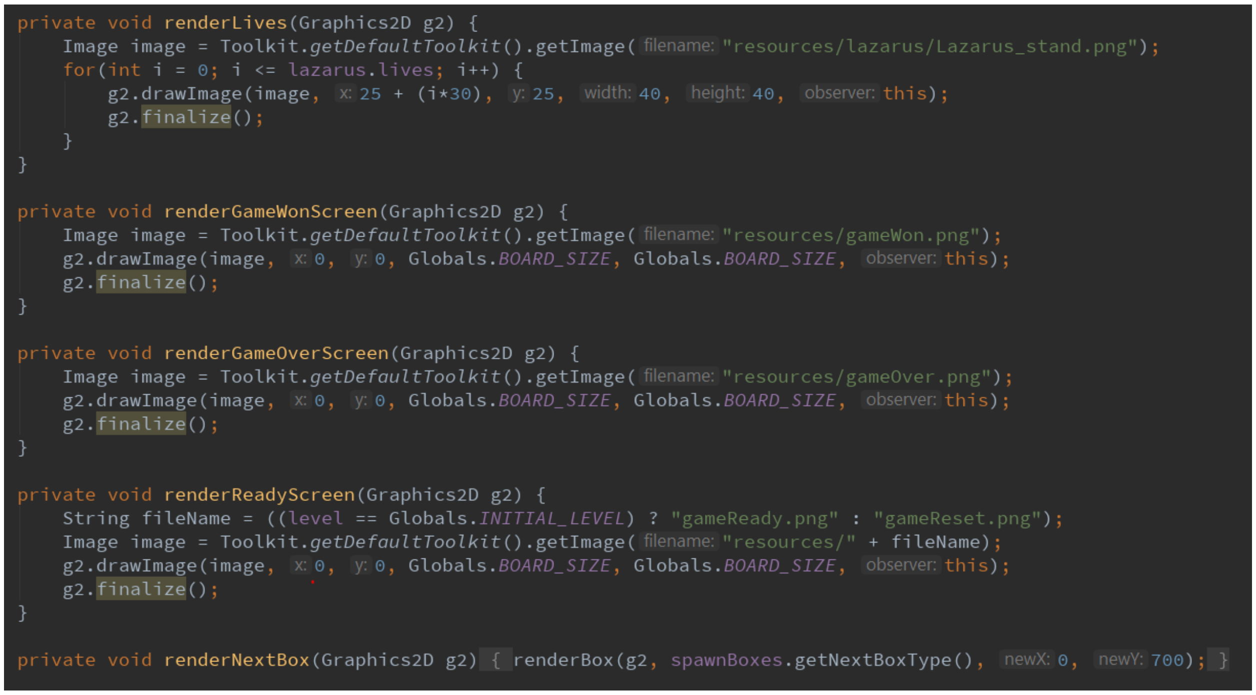


```java
/**
 * Resets/Restarts game to given level
 */
public void gameReset(int nextLevel) throws Exception {
    this.level = nextLevel;
    this.map = MapReader.readMap(level);
    this.gameState = Globals.GameState.READY;
    this.collision = new CollisionDetector(map);

    clearBoxes();

    findStartPosition();
    this.lazarus.resetPosition(startX, startY);
}

public void moveToNextLevel() throws Exception {
    int nextLevel = level + 1;
    if(nextLevel > Globals.FINAL_LEVEL) {
        setGameState(Globals.GameState.GAME_WON);
        return;
    }

    // Load next level
    gameReset(nextLevel);
}
```



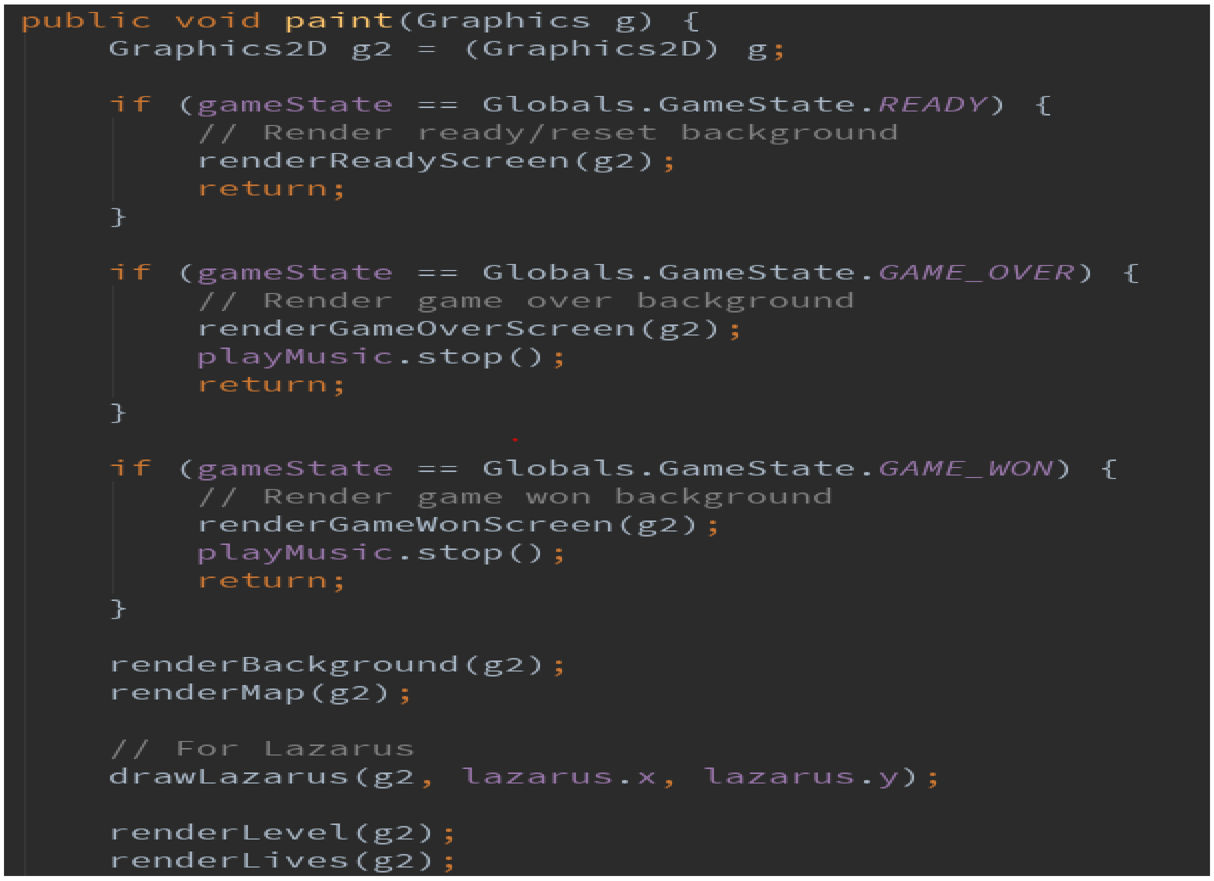


Some render functions as below:



```
private void renderLives(Graphics2D g2) {
    Image image = Toolkit.getDefaultToolkit().getImage( filename: "resources/lazarus/Lazarus_stand.png");
    for(int i = 0; i <= lazarus.lives; i++) {
        g2.drawImage(image,  x: 25 + (i*30),  y: 25,  width: 40,  height: 40,  observer: this);
        g2.finalize();
    }
}

private void renderGameWonScreen(Graphics2D g2) {
    Image image = Toolkit.getDefaultToolkit().getImage( filename: "resources/gameWon.png");
    g2.drawImage(image,  x: 0,  y: 0, Globals.BOARD_SIZE, Globals.BOARD_SIZE,  observer: this);
    g2.finalize();
}

private void renderGameOverScreen(Graphics2D g2) {
    Image image = Toolkit.getDefaultToolkit().getImage( filename: "resources/gameOver.png");
    g2.drawImage(image,  x: 0,  y: 0, Globals.BOARD_SIZE, Globals.BOARD_SIZE,  observer: this);
    g2.finalize();
}

private void renderReadyScreen(Graphics2D g2) {
    String fileName = ((level == Globals.INITIAL_LEVEL) ? "gameReady.png" : "gameReset.png");
    Image image = Toolkit.getDefaultToolkit().getImage( filename: "resources/" + fileName);
    g2.drawImage(image,  x: 0,  y: 0, Globals.BOARD_SIZE, Globals.BOARD_SIZE,  observer: this);
    g2.finalize();
}

private void renderNextBox(Graphics2D g2) { renderBox(g2, spawnBoxes.getNextBoxType(),  newX: 0,  newY: 700); }
```

paint() method: calls all the render functions and paints on the JFrame window.



```
public void paint(Graphics g) {
    Graphics2D g2 = (Graphics2D) g;

    if (gameState == Globals.GameState.READY) {
        // Render ready/reset background
        renderReadyScreen(g2);
        return;
    }

    if (gameState == Globals.GameState.GAME_OVER) {
        // Render game over background
        renderGameOverScreen(g2);
        playMusic.stop();
        return;
    }

    if (gameState == Globals.GameState.GAME_WON) {
        // Render game won background
        renderGameWonScreen(g2);
        playMusic.stop();
        return;
    }

    renderBackground(g2);
    renderMap(g2);

    // For Lazarus
    drawLazarus(g2, lazarus.x, lazarus.y);

    renderLevel(g2);
    renderLives(g2);
```

Game Over and Game Won screens:

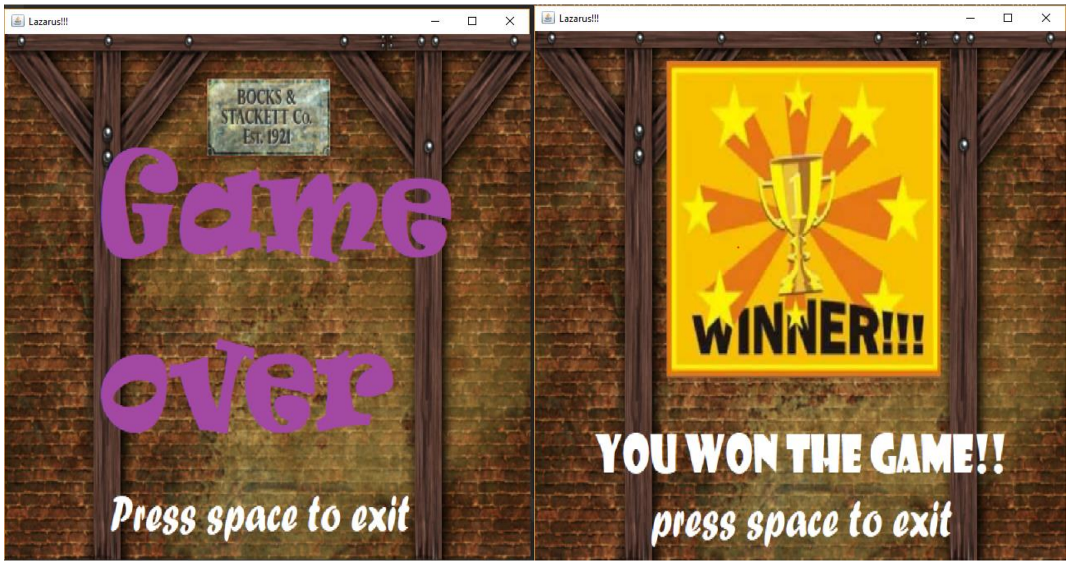


Chart Of Classes Shared Between Two Games:

One of the goals of these projects was to reuse the classes from the Tank game as much as possible in the Lazarus game. We used the same structure of Tank game and reused majority of the classes in the Tank game. The table shown below are the classes of the Tank game that were reused in the Lazarus game. Most of the classes have the same name.

Tank Game	**Lazarus**
AudioPlayer	AudioPlayer
CommonAPIs	CommonAPIs
Globals	Globals
MapReader	MapReader
CollisionDetector	CollisionDetector
KeysControl	KeysControl
Explosion	animateLeft animateRight animateJumpLeft aniamateJumpRight animateSquished
Driver	Driver
TankWorld	LazarusWorld

Self-Reflection on Software Design, Development and Reusability:

Throughout the entire project, there were many obstacles and challenges. How to construct the structure of the project, what could be the components of the game, how to connect all the components together were the big challenges. The reusability of the program was an excellent part among it, as reused classes are generic purpose classes and can be modified and reused again for future projects.

Conclusion:

To put it in a nutshell, both the games are almost finished. More specifically:

Tank game:

- Collision:
    - Collisions works as intended, the end result prevents the tank from colliding with the wall, another tank, and detects the collision between bullet and tank to reduce the health of the tank.
- Bullet:
    - Bullet moves in the direction of the tank, collides with the objects and updates the status accordingly.
- Simultaneous Tanks movement:
    - Both the tanks have the freedom to move/shoot at the same time.
- Minimap:
    - Minimap has been implemented, in case we decide to upgrade the game to have split screen features. As of now, the minimap is fully functional.

Lazarus game:

- Lazarus movement:
    - The lazarus only has two movements, that is left and right.
    - Detects whether or not the lazarus can move up or down the boxes depending on the height of the box. If it is within 1 box's height, the lazarus either moves up or down.
- Box Movement:
    - Spawning of boxes according to the Lazarus's position, and gradually descending to the lowest possible location.
    - Either breaks the box with lower priority, or stacks on top of the higher priority box.

Future Scope:

1. Movement of Tanks in all the direction (360 degree orientation).
2. Implement split screen features for the Tank game.
3. Lazarus animation fix. Currently, when the program renders the animation, it shows the small size of the Lazarus at the beginning of the animation.