OBJECT ORIENTED PROGRAMMING

FINAL PROJECT REPORT

“BATTLESHIP GAME”

By:

Ivandito Rakaputra

Lecturer:

Jude Joseph Lamug Martinez, MCS

School of Computing and Creative Arts
BINUS International University
2023

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating.

Binus International seriously regards all forms of plagiarism, cheating, and collusion as academic offences which may result in severe penalties, including loss/drop of marks, course/class discontinuity, and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality.

By signing this assignment, I understand, accept, and consent to Binus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student

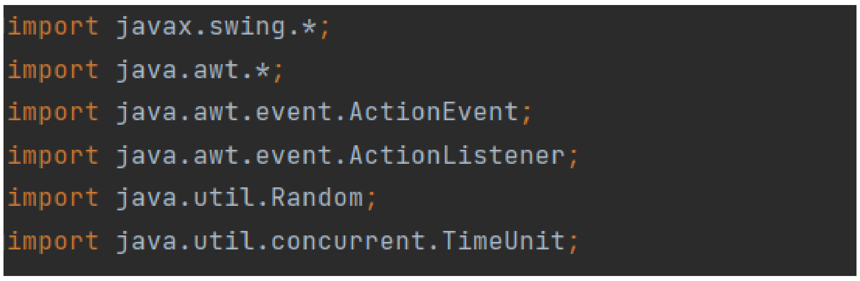
[illegible]

Ivandito Rakaputra

Table of Contents

I. Introduction

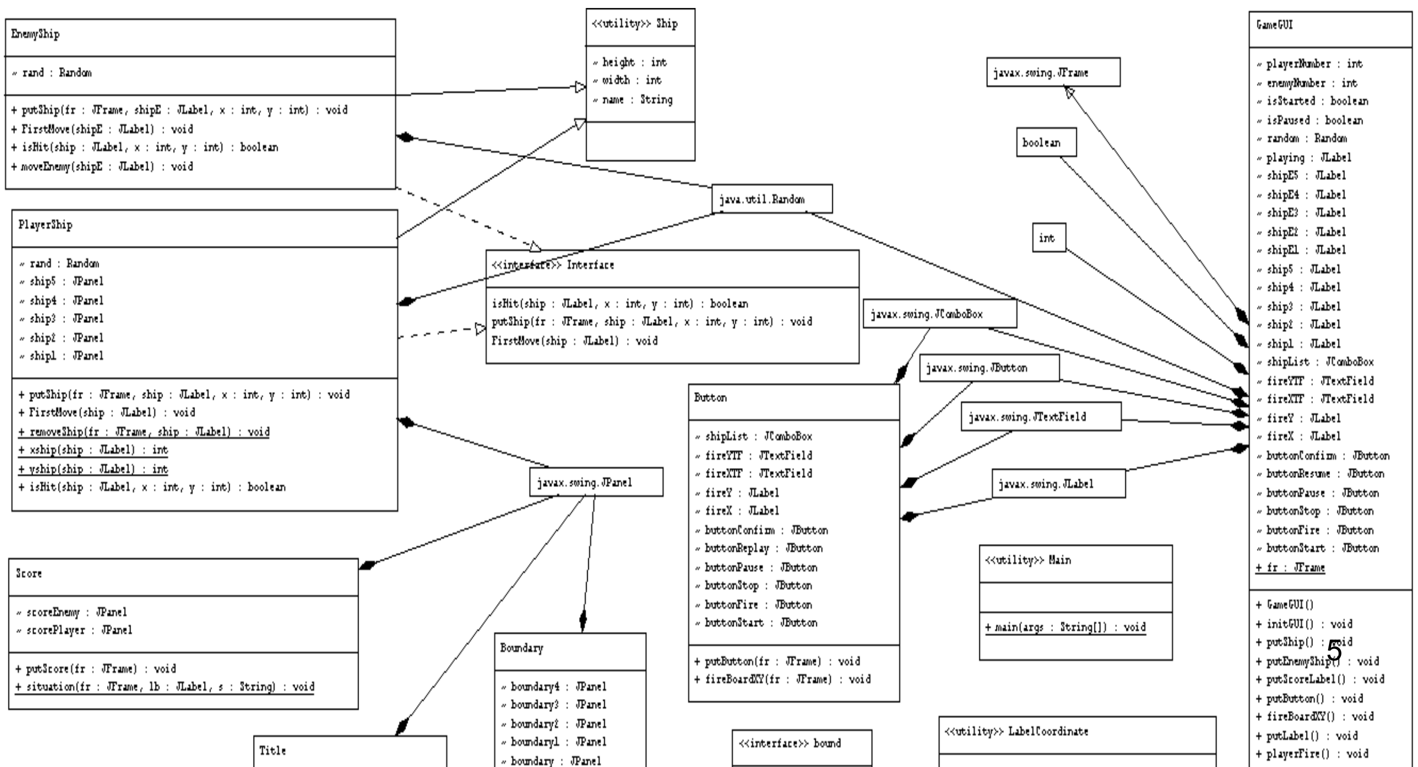
On this occasion, for the final project, I made a simple warship-themed game. This game is very simple and also made with a simple GUI. Libraries that I use are Java Swing, Java AWT (Abstract Window Toolkit) which I took, the Action Event and Action, Java Util random, and Java Util concurrent TimeUnit. Java Swing provides classes for java swing API such as JButton, JTextField, etc. Java AWT is used for creating the GUI. Java Util is used for providing classes that are crucial to the design of the Java programming language like to access files, file attributes, and file systems.



```java
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Random;
import java.util.concurrent.TimeUnit;
```

The explanation of this game is first of all, this game is a player vs bot. The game starts after pressing the "START" button on the main screen. Ships available on a screen are outlined by colour for the border. The border is denoted by x,y coordinates from 1 - 8. The number of our ship and enemy ships is 5, so 5 v 5. There is a "FIRE" button, which is used to attack enemies. There is an input for X and Y, which we use to attack the enemy according to their coordinates. Enemy ships can move randomly and change every 4 seconds, making it difficult for players to attack enemies. At the same time, after 4 seconds, the enemy ship also attacked the player ship randomly. Then after filling in the coordinates quickly, there is a "CONFIRM" button to attack. After we attack, at the same time our enemy also attacks us. There is a chance where the enemy can hit our ship or not and if the enemy hits us, our ship loses one. The same thing when it can hit an enemy ship, then one enemy ship is also missing. Then there is the "STOP" button as well, which is what it means to stop the game then when the player presses the "START" button again, the game will start all over again. Then there is also the "PAUSE" button and it does the same thing as the "STOP" button, only it doesn't reset the game, it just pauses the game for a while. When the player presses the "RESUME" button, the game will run again without starting over. The game is very simple, spend enemy ships or players spend with the enemy. If we win, the game returns to the "START" button. However, if we lose, the words "Game Over" appear and the player can only close the GUI screen, which means stopping the code and can start the game from the beginning again when running the code.

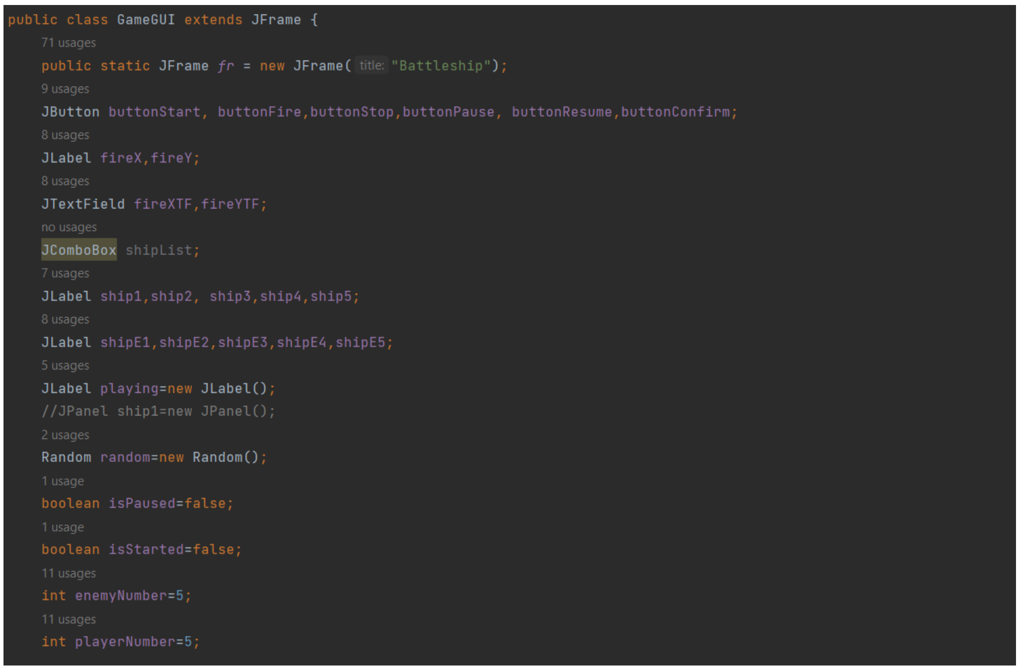
II. Class Diagram



III. Solutions Design

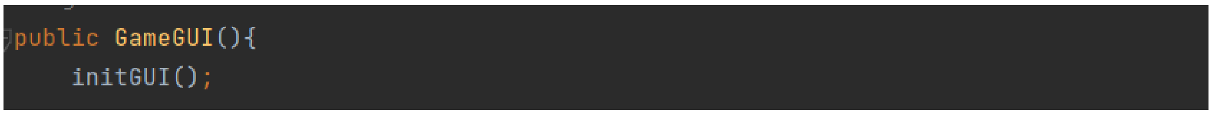
In making this project, there are several files that I use and will show you, for what these files help to make this project. Files involved to make this project are:

GameGUI.Java



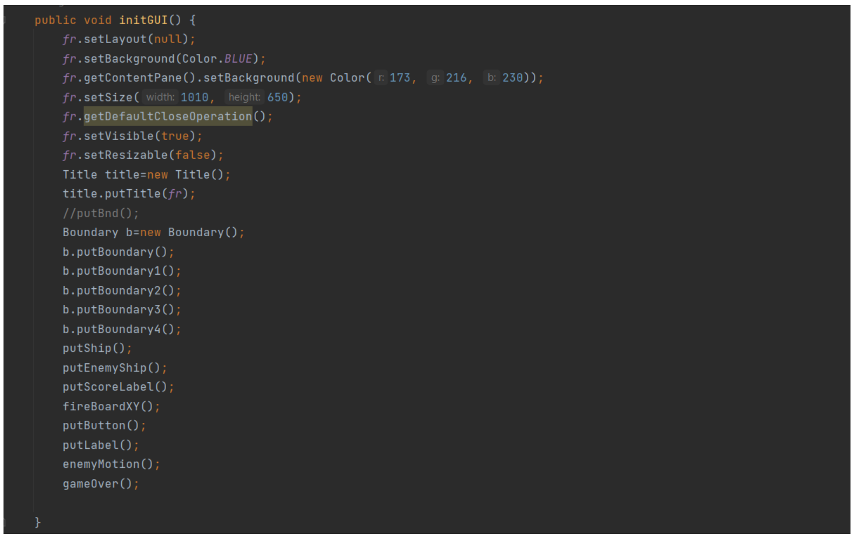
```java
public class GameGUI extends JFrame {
    71 usages
    public static JFrame fr = new JFrame( title: "Battleship");
    9 usages
    JButton buttonStart, buttonFire,buttonStop,buttonPause, buttonResume,buttonConfirm;
    8 usages
    JLabel fireX,fireY;
    8 usages
    JTextField fireXTF,fireYTF;
    no usages
    JComboBox shipList;
    7 usages
    JLabel ship1,ship2, ship3,ship4,ship5;
    8 usages
    JLabel shipE1,shipE2,shipE3,shipE4,shipE5;
    5 usages
    JLabel playing=new JLabel();
    //JPanel ship1=new JPanel();
    2 usages
    Random random=new Random();
    1 usage
    boolean isPaused=false;
    1 usage
    boolean isStarted=false;
    11 usages
    int enemyNumber=5;
    11 usages
    int playerNumber=5;
```

Starting from the GameGUI method which is responsible for creating and managing the GUI for the game.



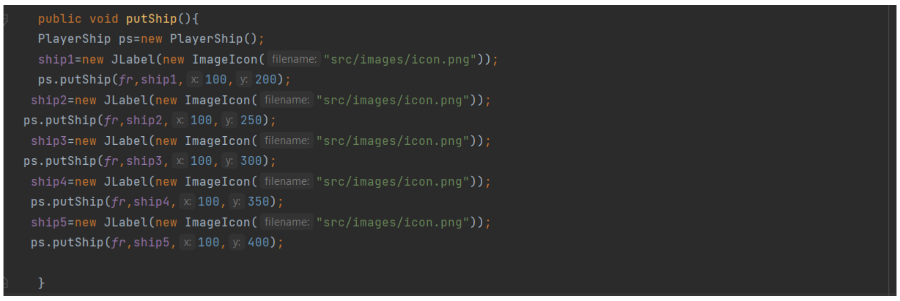
```java
public GameGUI(){
    initGUI();
```

Then it initialises the GUI components by calling the initGUI() method.



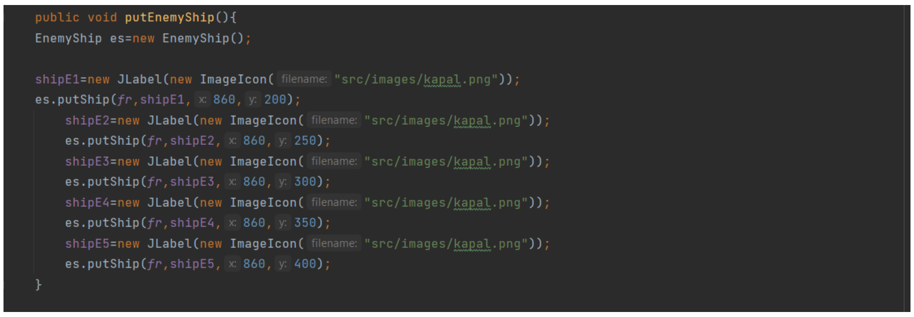
```java
public void initGUI() {
    fr.setLayout(null);
    fr.setBackground(Color.BLUE);
    fr.getContentPane().setBackground(new Color( r: 173,  g: 216,  b: 230));
    fr.setSize( width: 1010,  height: 650);
    fr.getDefaultCloseOperation();
    fr.setVisible(true);
    fr.setResizable(false);
    Title title=new Title();
    title.putTitle(fr);
    //putBnd();
    Boundary b=new Boundary();
    b.putBoundary();
    b.putBoundary1();
    b.putBoundary2();
    b.putBoundary3();
    b.putBoundary4();
    putShip();
    putEnemyShip();
    putScoreLabel();
    fireBoardXY();
    putButton();
    putLabel();
    enemyMotion();
    gameOver();

}
```

These are use for setting up the main JFrame window by configuring its layout, size, title, background colour, visibility, and other properties.

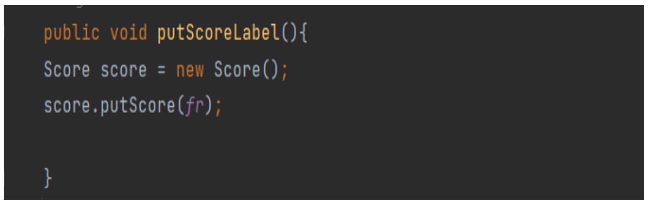


```java
 public void putShip(){
 PlayerShip ps=new PlayerShip();
 ship1=new JLabel(new ImageIcon( filename: "src/images/icon.png"));
 ps.putShip(fr,ship1, x: 100, y: 200);
ship2=new JLabel(new ImageIcon( filename: "src/images/icon.png"));
ps.putShip(fr,ship2, x: 100, y: 250);
ship3=new JLabel(new ImageIcon( filename: "src/images/icon.png"));
ps.putShip(fr,ship3, x: 100, y: 300);
ship4=new JLabel(new ImageIcon( filename: "src/images/icon.png"));
ps.putShip(fr,ship4, x: 100, y: 350);
ship5=new JLabel(new ImageIcon( filename: "src/images/icon.png"));
ps.putShip(fr,ship5, x: 100, y: 400);

 }
```

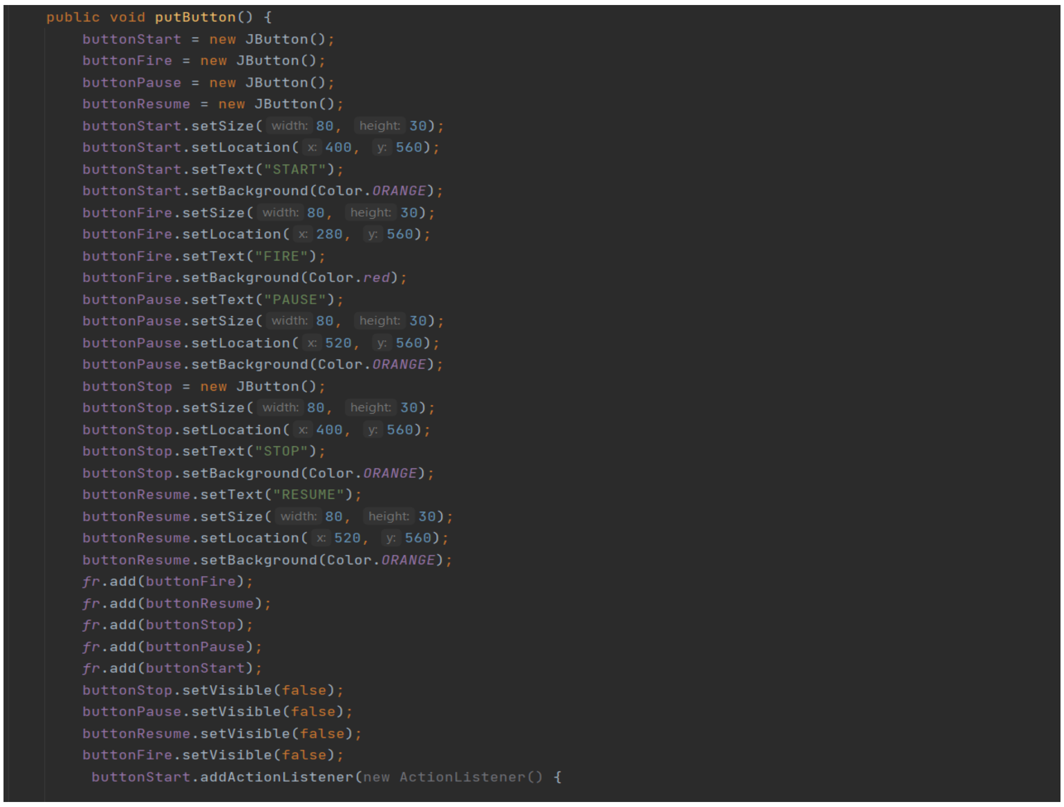
These are responsible for adding the Player Ships which call the PNG files.


```java
public void putEnemyShip(){
EnemyShip es=new EnemyShip();

shipE1=new JLabel(new ImageIcon( filename: "src/images/kapal.png"));
es.putShip(fr,shipE1, x: 860, y: 200);
    shipE2=new JLabel(new ImageIcon( filename: "src/images/kapal.png"));
    es.putShip(fr,shipE2, x: 860, y: 250);
    shipE3=new JLabel(new ImageIcon( filename: "src/images/kapal.png"));
    es.putShip(fr,shipE3, x: 860, y: 300);
    shipE4=new JLabel(new ImageIcon( filename: "src/images/kapal.png"));
    es.putShip(fr,shipE4, x: 860, y: 350);
    shipE5=new JLabel(new ImageIcon( filename: "src/images/kapal.png"));
    es.putShip(fr,shipE5, x: 860, y: 400);
}
```

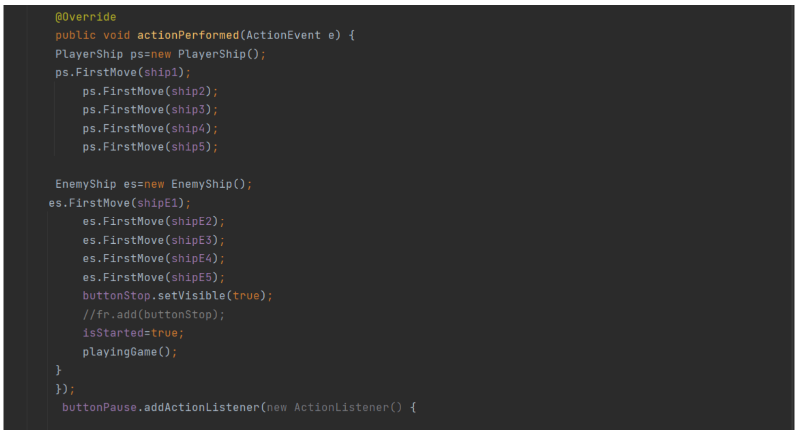
The same goes to the Enemies Ship, which I also call the PNG files.


```java
public void putScoreLabel(){
Score score = new Score();
score.putScore(fr);

}
```

These are responsible for adding a score label to the frame by creating an instance of the Score class and calling its putScore() method.

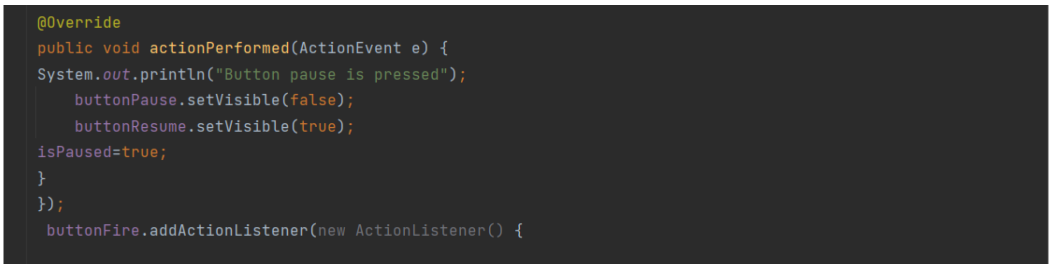

```java
public void putButton() {
    buttonStart = new JButton();
    buttonFire = new JButton();
    buttonPause = new JButton();
    buttonResume = new JButton();
    buttonStart.setSize( width: 80,  height: 30);
    buttonStart.setLocation( x: 400,  y: 560);
    buttonStart.setText("START");
    buttonStart.setBackground(Color.ORANGE);
    buttonFire.setSize( width: 80,  height: 30);
    buttonFire.setLocation( x: 280,  y: 560);
    buttonFire.setText("FIRE");
    buttonFire.setBackground(Color.red);
    buttonPause.setText("PAUSE");
    buttonPause.setSize( width: 80,  height: 30);
    buttonPause.setLocation( x: 520,  y: 560);
    buttonPause.setBackground(Color.ORANGE);
    buttonStop = new JButton();
    buttonStop.setSize( width: 80,  height: 30);
    buttonStop.setLocation( x: 400,  y: 560);
    buttonStop.setText("STOP");
    buttonStop.setBackground(Color.ORANGE);
    buttonResume.setText("RESUME");
    buttonResume.setSize( width: 80,  height: 30);
    buttonResume.setLocation( x: 520,  y: 560);
    buttonResume.setBackground(Color.ORANGE);
    fr.add(buttonFire);
    fr.add(buttonResume);
    fr.add(buttonStop);
    fr.add(buttonPause);
    fr.add(buttonStart);
    buttonStop.setVisible(false);
    buttonPause.setVisible(false);
    buttonResume.setVisible(false);
    buttonFire.setVisible(false);
     buttonStart.addActionListener(new ActionListener() {
```

These are for adds buttons to the frame, including "START", "FIRE", "PAUSE", "RESUME", and "STOP" buttons.



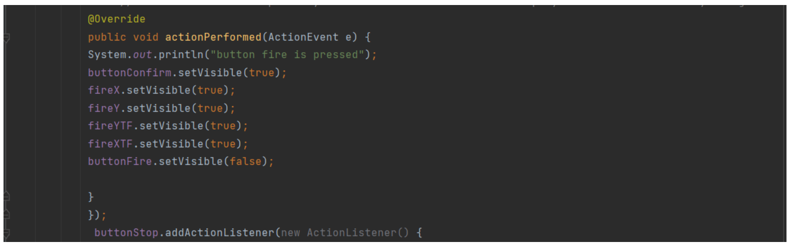
```
 @Override
 public void actionPerformed(ActionEvent e) {
 PlayerShip ps=new PlayerShip();
 ps.FirstMove(ship1);
     ps.FirstMove(ship2);
     ps.FirstMove(ship3);
     ps.FirstMove(ship4);
     ps.FirstMove(ship5);

 EnemyShip es=new EnemyShip();
es.FirstMove(shipE1);
     es.FirstMove(shipE2);
     es.FirstMove(shipE3);
     es.FirstMove(shipE4);
     es.FirstMove(shipE5);
     buttonStop.setVisible(true);
     //fr.add(buttonStop);
     isStarted=true;
     playingGame();
 }
 });
  buttonPause.addActionListener(new ActionListener() {
```

When the button "start" is performed, the game starts. The start button change to "stop"



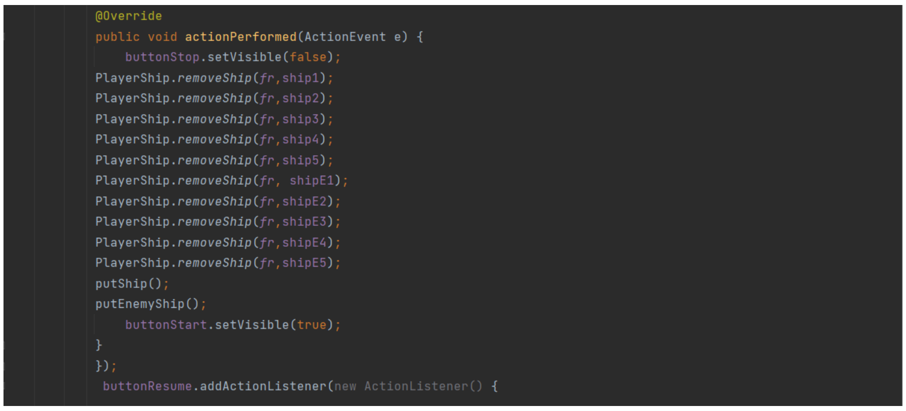
```
@Override
public void actionPerformed(ActionEvent e) {
System.out.println("Button pause is pressed");
    buttonPause.setVisible(false);
    buttonResume.setVisible(true);
isPaused=true;
}
});
 buttonFire.addActionListener(new ActionListener() {
```

When the button "pause" is pressed, the game pauses until the players press the "resume" button.

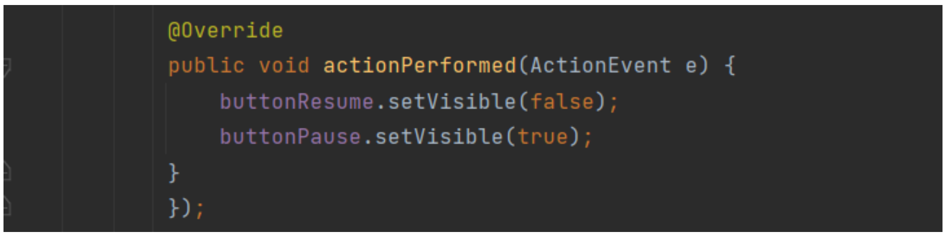


```
@Override
public void actionPerformed(ActionEvent e) {
System.out.println("button fire is pressed");
buttonConfirm.setVisible(true);
fireX.setVisible(true);
fireY.setVisible(true);
fireYTF.setVisible(true);
fireXTF.setVisible(true);
buttonFire.setVisible(false);

}
});
 buttonStop.addActionListener(new ActionListener() {
```

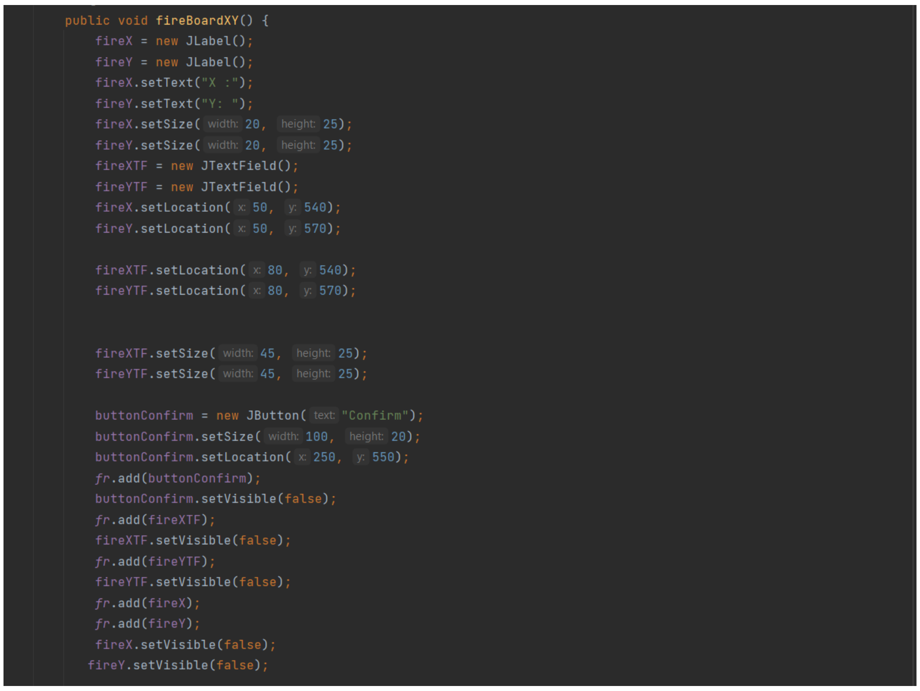
When the button "fire" is pressed, it calls the fireBoardXY so the player can attack the enemy using the coordinate.


```java
@Override
public void actionPerformed(ActionEvent e) {
    buttonStop.setVisible(false);
PlayerShip.removeShip(fr,ship1);
PlayerShip.removeShip(fr,ship2);
PlayerShip.removeShip(fr,ship3);
PlayerShip.removeShip(fr,ship4);
PlayerShip.removeShip(fr,ship5);
PlayerShip.removeShip(fr, shipE1);
PlayerShip.removeShip(fr,shipE2);
PlayerShip.removeShip(fr,shipE3);
PlayerShip.removeShip(fr,shipE4);
PlayerShip.removeShip(fr,shipE5);
putShip();
putEnemyShip();
    buttonStart.setVisible(true);
}
});
 buttonResume.addActionListener(new ActionListener() {
```

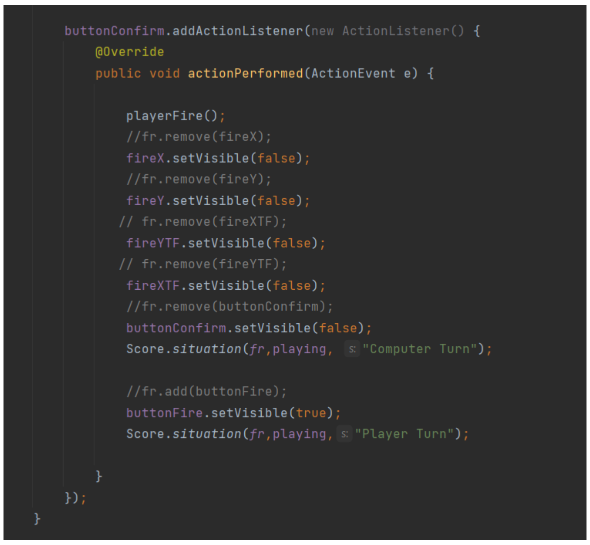
When the "stop" button is pressed, it starts the game from the beginning.


```java
@Override
public void actionPerformed(ActionEvent e) {
    buttonResume.setVisible(false);
    buttonPause.setVisible(true);
}
});
```

It removes the "Resume" button from the frame and adds the "Pause" button instead.

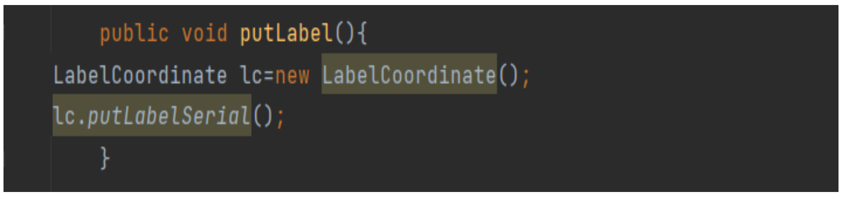

```java
public void fireBoardXY() {
    fireX = new JLabel();
    fireY = new JLabel();
    fireX.setText("X :");
    fireY.setText("Y: ");
    fireX.setSize( width: 20,  height: 25);
    fireY.setSize( width: 20,  height: 25);
    fireXTF = new JTextField();
    fireYTF = new JTextField();
    fireX.setLocation( x: 50,  y: 540);
    fireY.setLocation( x: 50,  y: 570);

    fireXTF.setLocation( x: 80,  y: 540);
    fireYTF.setLocation( x: 80,  y: 570);


    fireXTF.setSize( width: 45,  height: 25);
    fireYTF.setSize( width: 45,  height: 25);

    buttonConfirm = new JButton( text: "Confirm");
    buttonConfirm.setSize( width: 100,  height: 20);
    buttonConfirm.setLocation( x: 250,  y: 550);
    fr.add(buttonConfirm);
    buttonConfirm.setVisible(false);
    fr.add(fireXTF);
    fireXTF.setVisible(false);
    fr.add(fireYTF);
    fireYTF.setVisible(false);
    fr.add(fireX);
    fr.add(fireY);
    fireX.setVisible(false);
   fireY.setVisible(false);
```

This is triggered when the "FIRE" button is clicked. Later, it creates GUI components for entering the X and Y coordinates to fire at the enemy ship.



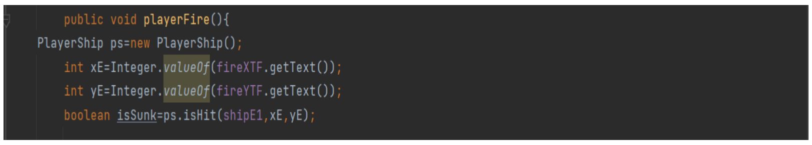
```java
    buttonConfirm.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {

            playerFire();
            //fr.remove(fireX);
            fireX.setVisible(false);
            //fr.remove(fireY);
            fireY.setVisible(false);
           // fr.remove(fireXTF);
            fireYTF.setVisible(false);
           // fr.remove(fireYTF);
            fireXTF.setVisible(false);
            //fr.remove(buttonConfirm);
            buttonConfirm.setVisible(false);
            Score.situation(fr,playing,  s: "Computer Turn");

            //fr.add(buttonFire);
            buttonFire.setVisible(true);
            Score.situation(fr,playing, s: "Player Turn");

        }
    });
}
```

After the "Confirm" button is pressed, it retrieves the entered coordinates, removes the GUI components, and calls playerFire() and enemyFire() methods to check for hits and update the game state accordingly.



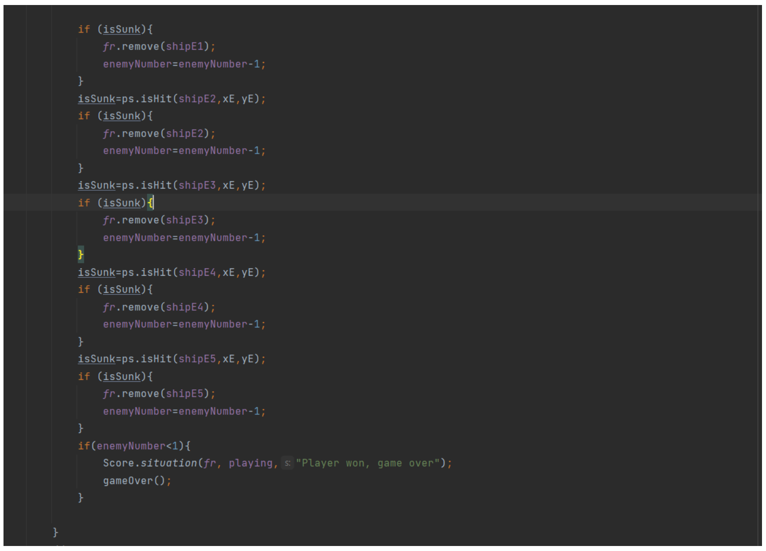
```java
    public void putLabel(){
LabelCoordinate lc=new LabelCoordinate();
lc.putLabelSerial();
    }
```

It adds labels for coordinate markers on the frame. Also, it calls the putLabelSerial() method of the LabelCoordinate class to place labels on the game board.



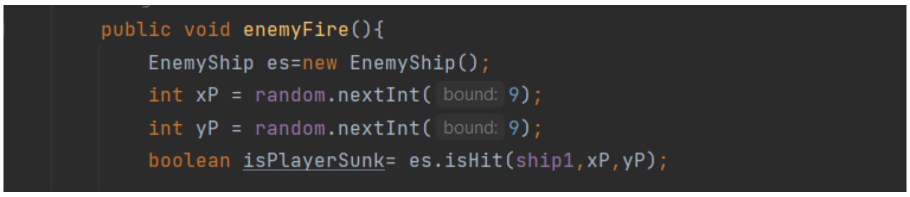
```java
    public void playerFire(){
PlayerShip ps=new PlayerShip();
    int xE=Integer.valueOf(fireXTF.getText());
    int yE=Integer.valueOf(fireYTF.getText());
    boolean isSunk=ps.isHit(shipE1,xE,yE);
```

These are for handling the fire action button. It retrieves the X and Y coordinates entered by the player from the fireXTF and fireYTF text fields.



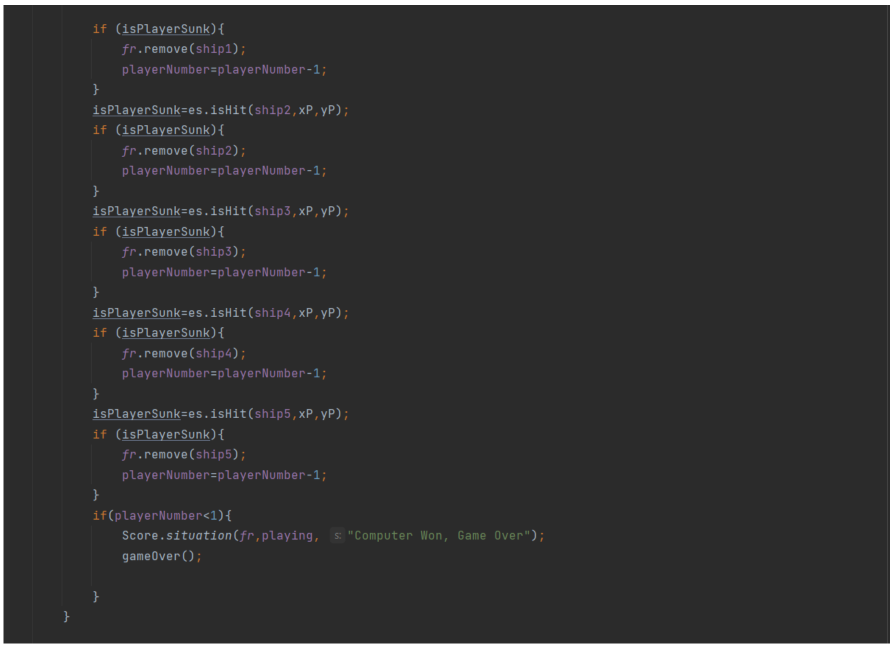
```
        if (isSunk){
            fr.remove(shipE1);
            enemyNumber=enemyNumber-1;
        }
        isSunk=ps.isHit(shipE2,xE,yE);
        if (isSunk){
            fr.remove(shipE2);
            enemyNumber=enemyNumber-1;
        }
        isSunk=ps.isHit(shipE3,xE,yE);
        if (isSunk){
            fr.remove(shipE3);
            enemyNumber=enemyNumber-1;
        }
        isSunk=ps.isHit(shipE4,xE,yE);
        if (isSunk){
            fr.remove(shipE4);
            enemyNumber=enemyNumber-1;
        }
        isSunk=ps.isHit(shipE5,xE,yE);
        if (isSunk){
            fr.remove(shipE5);
            enemyNumber=enemyNumber-1;
        }
        if(enemyNumber<1){
            Score.situation(fr, playing, s: "Player won, game over");
            gameOver();
        }

    }
```

It checks if the player's shots hit any enemy ships using the isEnemyHit() method of the PlayerShip class. The fr.remove methods are the reason the game is a bit buggy. The methods aren't optimal and it's heavy. If the player win, the terminal will say " Player won, game over"



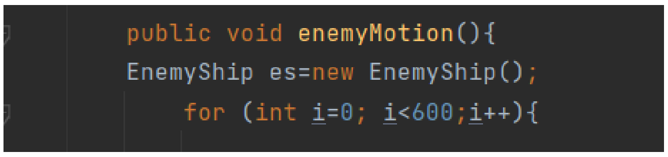
```
    public void enemyFire(){
        EnemyShip es=new EnemyShip();
        int xP = random.nextInt( bound: 9);
        int yP = random.nextInt( bound: 9);
        boolean isPlayerSunk= es.isHit(ship1,xP,yP);
```

These are for handling fire action.



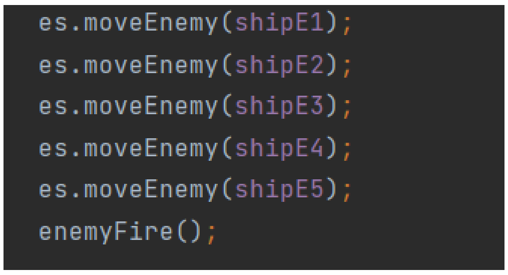
```
        if (isPlayerSunk){
            fr.remove(ship1);
            playerNumber=playerNumber-1;
        }
        isPlayerSunk=es.isHit(ship2,xP,yP);
        if (isPlayerSunk){
            fr.remove(ship2);
            playerNumber=playerNumber-1;
        }
        isPlayerSunk=es.isHit(ship3,xP,yP);
        if (isPlayerSunk){
            fr.remove(ship3);
            playerNumber=playerNumber-1;
        }
        isPlayerSunk=es.isHit(ship4,xP,yP);
        if (isPlayerSunk){
            fr.remove(ship4);
            playerNumber=playerNumber-1;
        }
        isPlayerSunk=es.isHit(ship5,xP,yP);
        if (isPlayerSunk){
            fr.remove(ship5);
            playerNumber=playerNumber-1;
        }
        if(playerNumber<1){
            Score.situation(fr,playing, s: "Computer Won, Game Over");
            gameOver();

        }
    }
```

It checks if the enemy's shots hit any player ships using the isPlayerHit() method of the EnemyShip class. The same case happened, The fr.remove methods are the reason the game is a bit buggy. If the computer wins, the terminal will print “Computer Won, Game Over”.



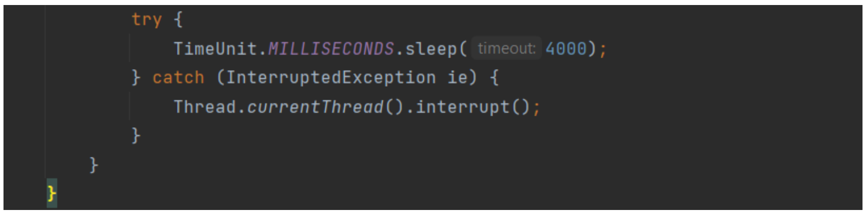
```
    public void enemyMotion(){
    EnemyShip es=new EnemyShip();
        for (int i=0; i<600;i++){
```

This handles the motion of enemy ships.



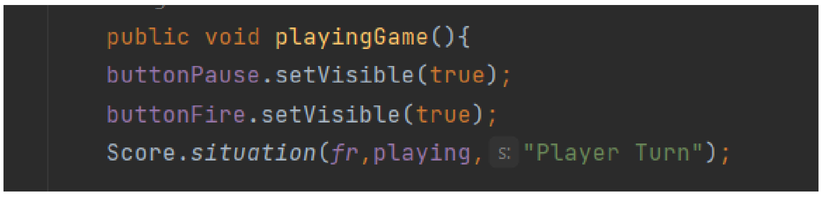
```
es.moveEnemy(shipE1);
es.moveEnemy(shipE2);
es.moveEnemy(shipE3);
es.moveEnemy(shipE4);
es.moveEnemy(shipE5);
enemyFire();
```

It repeatedly calls the moveEnemy() method of the EnemyShip class to move each enemy ship and at the same time, fires the player ship randomly.



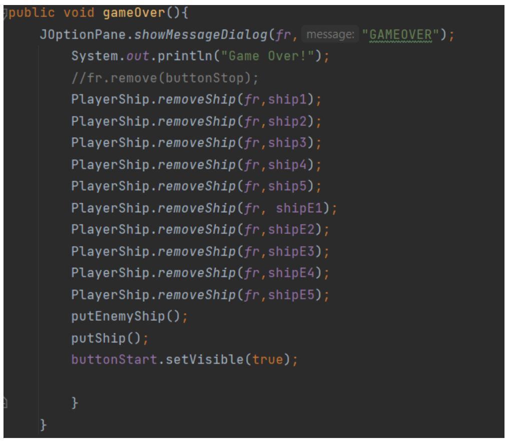
```
            try {
                TimeUnit.MILLISECONDS.sleep( timeout: 4000);
            } catch (InterruptedException ie) {
                Thread.currentThread().interrupt();
            }
        }
    }
```

After each ship is moved, there is a delay of 4000 milliseconds (4 seconds) before moving the next ship.



```
    public void playingGame(){
    buttonPause.setVisible(true);
    buttonFire.setVisible(true);
    Score.situation(fr,playing, s: "Player Turn");
```

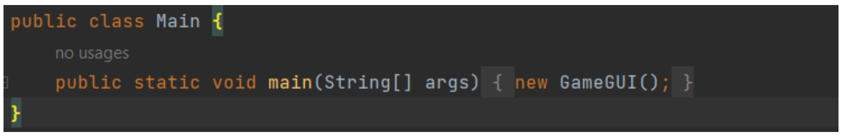
When the start button is pressed.



```
public void gameOver(){
    JOptionPane.showMessageDialog(fr, message: "GAMEOVER");
        System.out.println("Game Over!");
        //fr.remove(buttonStop);
        PlayerShip.removeShip(fr,ship1);
        PlayerShip.removeShip(fr,ship2);
        PlayerShip.removeShip(fr,ship3);
        PlayerShip.removeShip(fr,ship4);
        PlayerShip.removeShip(fr,ship5);
        PlayerShip.removeShip(fr, shipE1);
        PlayerShip.removeShip(fr,shipE2);
        PlayerShip.removeShip(fr,shipE3);
        PlayerShip.removeShip(fr,shipE4);
        PlayerShip.removeShip(fr,shipE5);
        putEnemyShip();
        putShip();
        buttonStart.setVisible(true);

        }
    }
```

This is responsible for handling the game over. It displays the message "GAMEOVER"

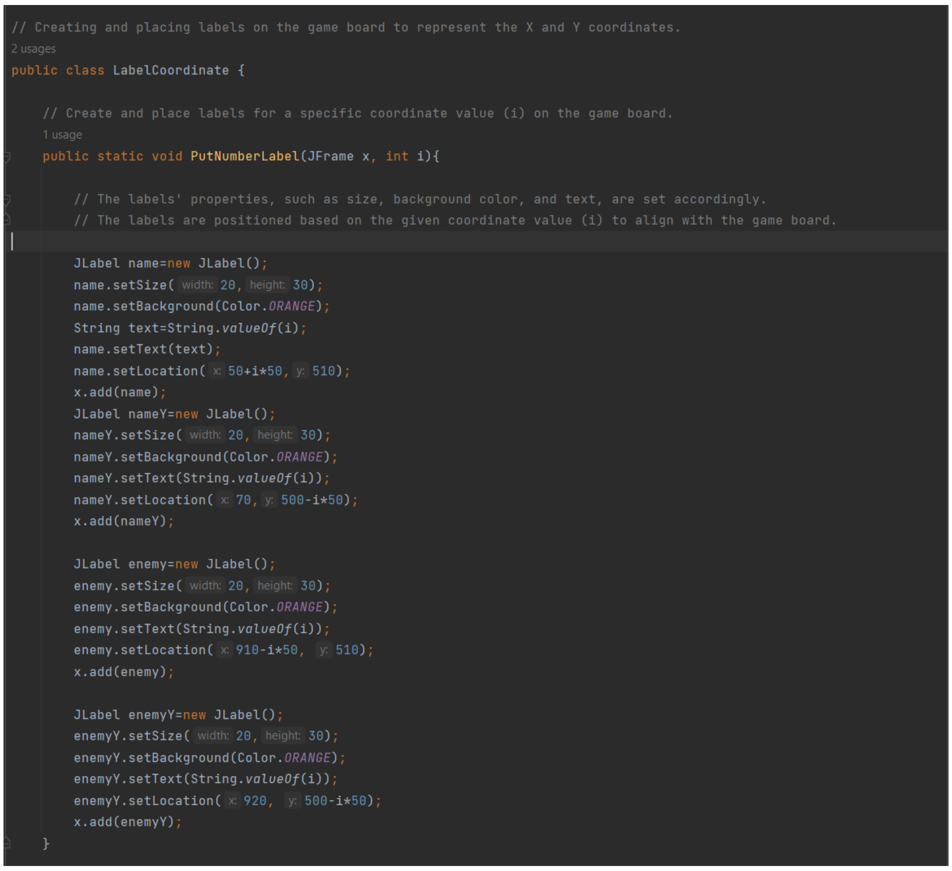
Main.java



```
public class Main {
    no usages
    public static void main(String[] args) { new GameGUI(); }
}
```

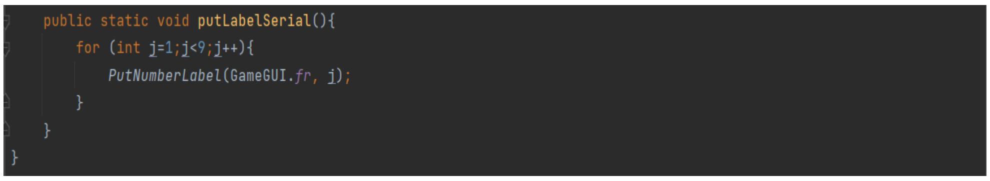
This is the main method that is automatically executed when the program starts.

LabelCoordinate.java



```java
// Creating and placing labels on the game board to represent the X and Y coordinates.
2 usages
public class LabelCoordinate {

    // Create and place labels for a specific coordinate value (i) on the game board.
    1 usage
    public static void PutNumberLabel(JFrame x, int i){

        // The labels' properties, such as size, background color, and text, are set accordingly.
        // The labels are positioned based on the given coordinate value (i) to align with the game board.

        JLabel name=new JLabel();
        name.setSize( width: 20, height: 30);
        name.setBackground(Color.ORANGE);
        String text=String.valueOf(i);
        name.setText(text);
        name.setLocation( x: 50+i*50, y: 510);
        x.add(name);
        JLabel nameY=new JLabel();
        nameY.setSize( width: 20, height: 30);
        nameY.setBackground(Color.ORANGE);
        nameY.setText(String.valueOf(i));
        nameY.setLocation( x: 70, y: 500-i*50);
        x.add(nameY);

        JLabel enemy=new JLabel();
        enemy.setSize( width: 20, height: 30);
        enemy.setBackground(Color.ORANGE);
        enemy.setText(String.valueOf(i));
        enemy.setLocation( x: 910-i*50,  y: 510);
        x.add(enemy);

        JLabel enemyY=new JLabel();
        enemyY.setSize( width: 20, height: 30);
        enemyY.setBackground(Color.ORANGE);
        enemyY.setText(String.valueOf(i));
        enemyY.setLocation( x: 920,  y: 500-i*50);
        x.add(enemyY);
    }
```

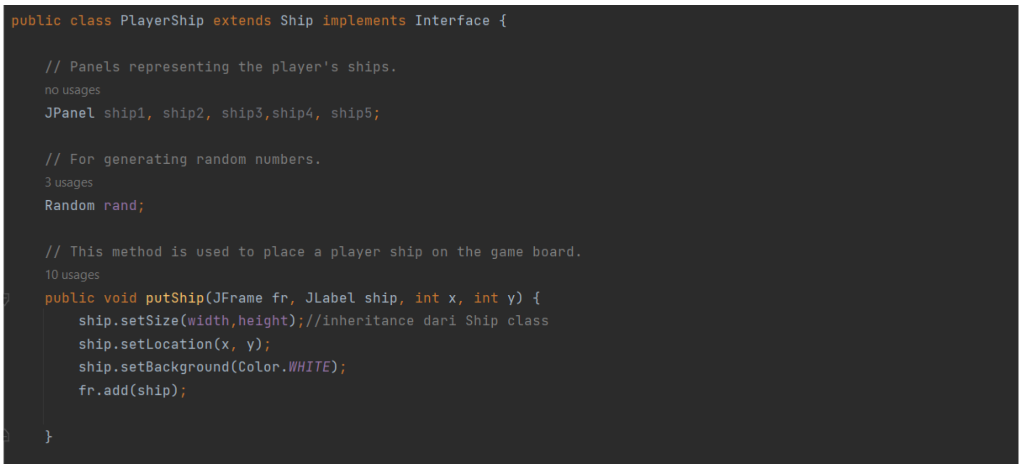
The public class LabelCoordiante are responsible for creating and placing labels on the game board to represent the X and Y coordinates. The PutNumberLabel is to create and place labels for a specific coordinate value (i) on the game board. The labels' properties, such as size, background colour, and text, are set accordingly. The labels are positioned based on the given coordinate value (i) to align with the game board.



```java
    public static void putLabelSerial(){
        for (int j=1;j<9;j++){
            PutNumberLabel(GameGUI.fr, j);
        }
    }
}
```

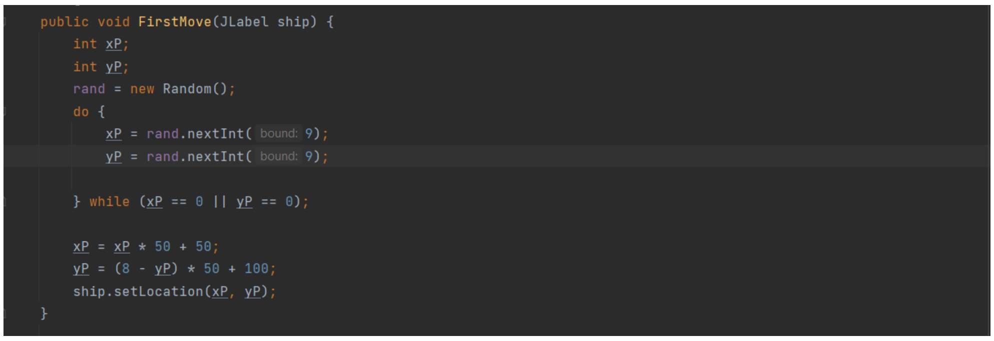
Used to generate and place labels for the serial numbers of the coordinates.

PlayerShip.java



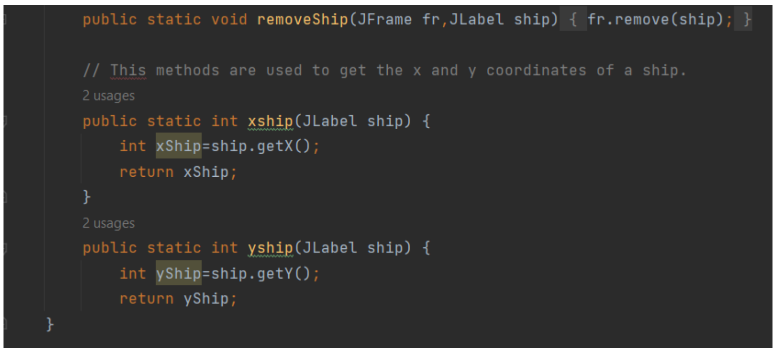
```java
public class PlayerShip extends Ship implements Interface {

    // Panels representing the player's ships.
    no usages
    JPanel ship1, ship2, ship3,ship4, ship5;

    // For generating random numbers.
    3 usages
    Random rand;

    // This method is used to place a player ship on the game board.
    10 usages
    public void putShip(JFrame fr, JLabel ship, int x, int y) {
        ship.setSize(width,height);//inheritance dari Ship class
        ship.setLocation(x, y);
        ship.setBackground(Color.WHITE);
        fr.add(ship);

    }
```

This method represents the player's ships in the game. It contains methods for placing and managing player ships on the game board.



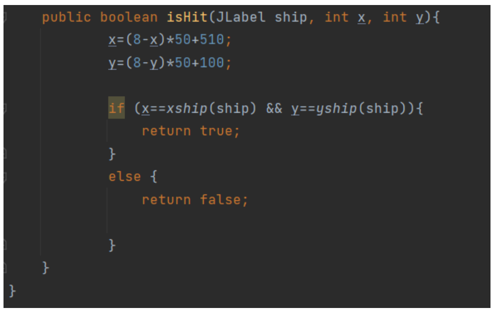
```java
    public void FirstMove(JLabel ship) {
        int xP;
        int yP;
        rand = new Random();
        do {
            xP = rand.nextInt( bound: 9);
            yP = rand.nextInt( bound: 9);

        } while (xP == 0 || yP == 0);

        xP = xP * 50 + 50;
        yP = (8 - yP) * 50 + 100;
        ship.setLocation(xP, yP);
    }
```

This method is used to set the initial position of the player ship.



```java
        public static void removeShip(JFrame fr,JLabel ship) { fr.remove(ship); }

        // This methods are used to get the x and y coordinates of a ship.
        2 usages
        public static int xship(JLabel ship) {
            int xShip=ship.getX();
            return xShip;
        }
        2 usages
        public static int yship(JLabel ship) {
            int yShip=ship.getY();
            return yShip;
    }
```

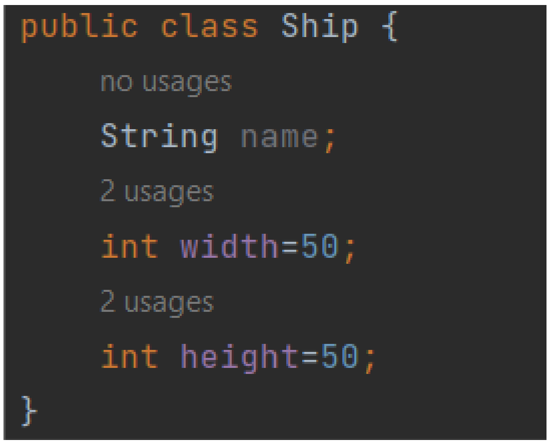
This method is used to remove a ship from the game frame.



```
    public boolean isHit(JLabel ship, int x, int y){
            x=(8-x)*50+510;
            y=(8-y)*50+100;

            if (x==xship(ship) && y==yship(ship)){
                return true;
            }
            else {
                return false;

            }
    }
}
```

This method is used to check if an enemy ship has been hit by the player. If the coordinates match the position of the ship, it indicates a hit and truth is returned. Otherwise, it indicates a miss and false is returned.

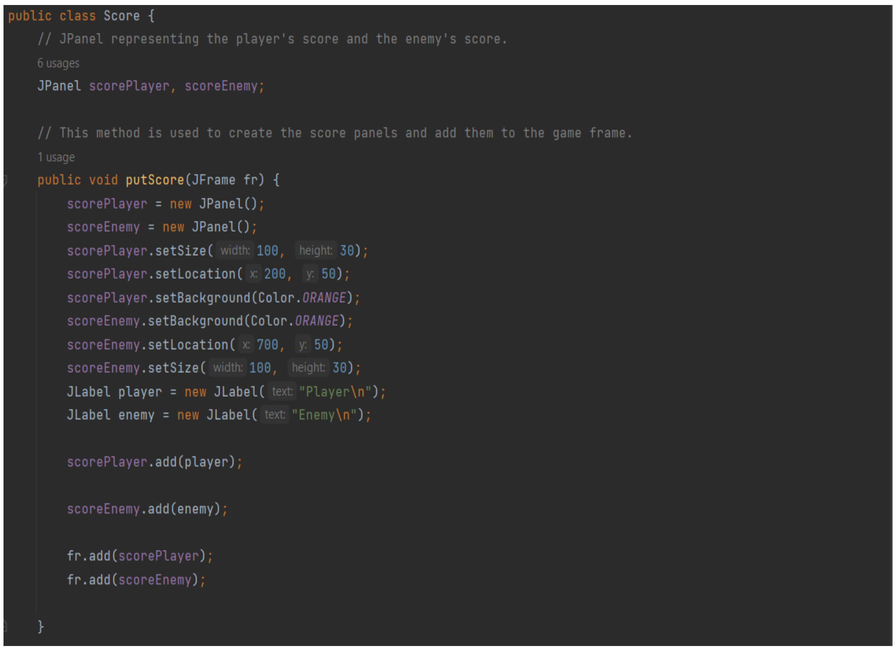
Ship.java



```
public class Ship {
    no usages
    String name;
    2 usages
    int width=50;
    2 usages
    int height=50;
}
```

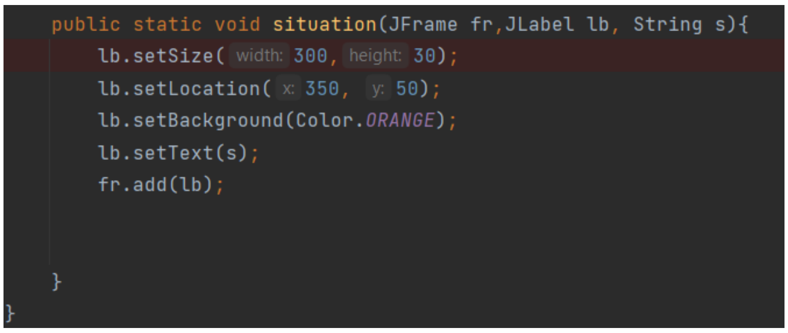
This method is to represent a blueprint for creating objects of type "Ship." in GameGUI.java.

Score.java



```java
public class Score {
    // JPanel representing the player's score and the enemy's score.
    6 usages
    JPanel scorePlayer, scoreEnemy;

    // This method is used to create the score panels and add them to the game frame.
    1 usage
    public void putScore(JFrame fr) {
        scorePlayer = new JPanel();
        scoreEnemy = new JPanel();
        scorePlayer.setSize( width: 100,  height: 30);
        scorePlayer.setLocation( x: 200,  y: 50);
        scorePlayer.setBackground(Color.ORANGE);
        scoreEnemy.setBackground(Color.ORANGE);
        scoreEnemy.setLocation( x: 700,  y: 50);
        scoreEnemy.setSize( width: 100,  height: 30);
        JLabel player = new JLabel( text: "Player\n");
        JLabel enemy = new JLabel( text: "Enemy\n");

        scorePlayer.add(player);

        scoreEnemy.add(enemy);

        fr.add(scorePlayer);
        fr.add(scoreEnemy);

    }
```

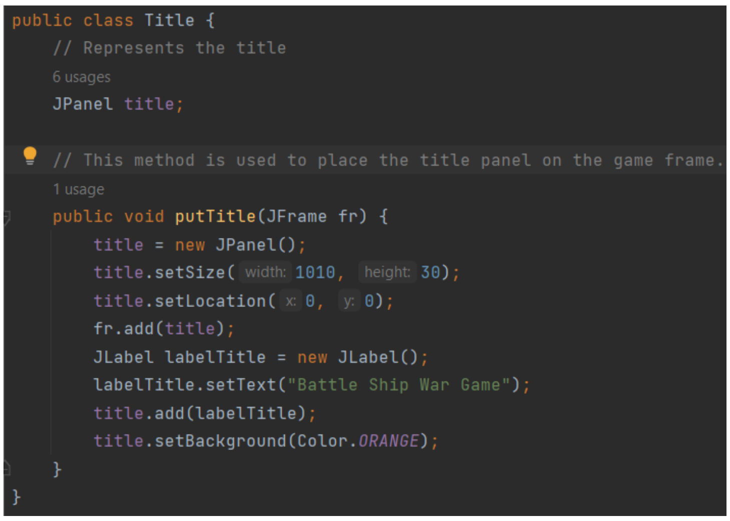
This method is for managing and displaying the scores of the player and the enemy in the game.



```java
    public static void situation(JFrame fr,JLabel lb, String s){
        lb.setSize( width: 300, height: 30);
        lb.setLocation( x: 350,  y: 50);
        lb.setBackground(Color.ORANGE);
        lb.setText(s);
        fr.add(lb);


    }
}
```

This is used to update the situation label, which provides information about the game status or current situation.

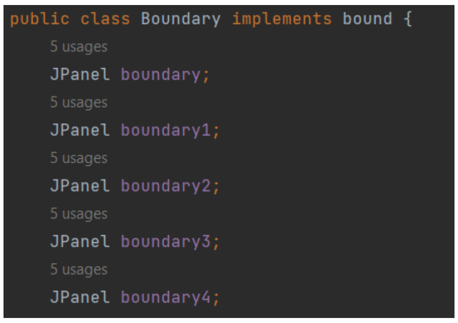
Title.java



```
public class Title {
    // Represents the title
    6 usages
    JPanel title;

    // This method is used to place the title panel on the game frame.
    1 usage
    public void putTitle(JFrame fr) {
        title = new JPanel();
        title.setSize( width: 1010,  height: 30);
        title.setLocation( x: 0,  y: 0);
        fr.add(title);
        JLabel labelTitle = new JLabel();
        labelTitle.setText("Battle Ship War Game");
        title.add(labelTitle);
        title.setBackground(Color.ORANGE);
    }
}
```

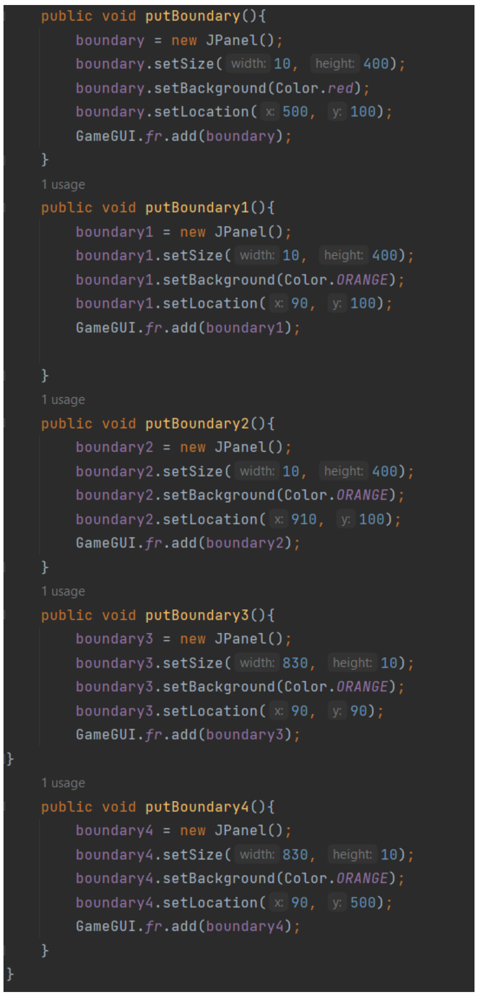
This are responsible for displaying the title of the game at the top of the game frame.

Boundary.java



```
public class Boundary implements bound {
    5 usages
    JPanel boundary;
    5 usages
    JPanel boundary1;
    5 usages
    JPanel boundary2;
    5 usages
    JPanel boundary3;
    5 usages
    JPanel boundary4;
```

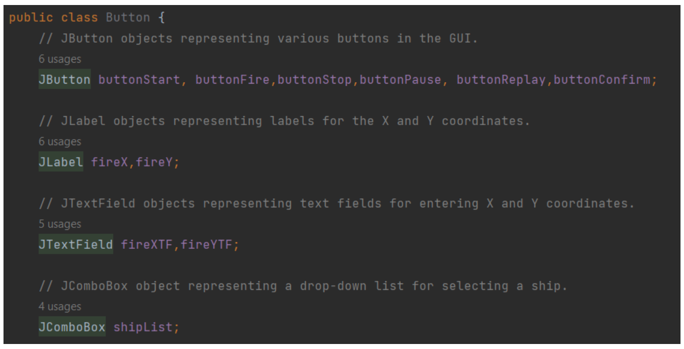
This method is used to create the boundary.



```
    public void putBoundary(){
        boundary = new JPanel();
        boundary.setSize( width: 10,  height: 400);
        boundary.setBackground(Color.red);
        boundary.setLocation( x: 500,  y: 100);
        GameGUI.fr.add(boundary);
    }
    1 usage
    public void putBoundary1(){
        boundary1 = new JPanel();
        boundary1.setSize( width: 10,  height: 400);
        boundary1.setBackground(Color.ORANGE);
        boundary1.setLocation( x: 90,  y: 100);
        GameGUI.fr.add(boundary1);

    }
    1 usage
    public void putBoundary2(){
        boundary2 = new JPanel();
        boundary2.setSize( width: 10,  height: 400);
        boundary2.setBackground(Color.ORANGE);
        boundary2.setLocation( x: 910,  y: 100);
        GameGUI.fr.add(boundary2);
    }
    1 usage
    public void putBoundary3(){
        boundary3 = new JPanel();
        boundary3.setSize( width: 830,  height: 10);
        boundary3.setBackground(Color.ORANGE);
        boundary3.setLocation( x: 90,  y: 90);
        GameGUI.fr.add(boundary3);
}
    1 usage
    public void putBoundary4(){
        boundary4 = new JPanel();
        boundary4.setSize( width: 830,  height: 10);
        boundary4.setBackground(Color.ORANGE);
        boundary4.setLocation( x: 90,  y: 500);
        GameGUI.fr.add(boundary4);
    }
}
```

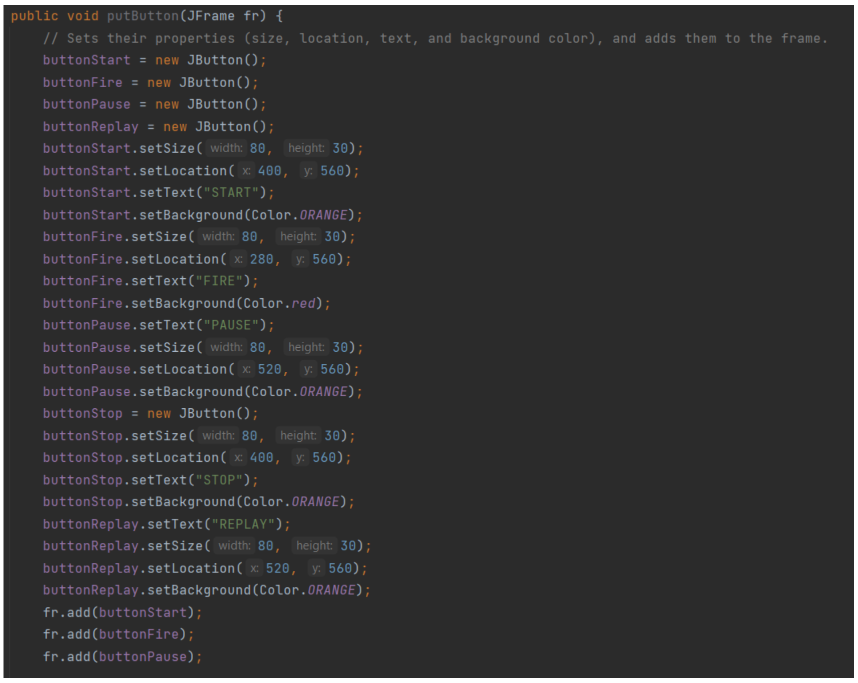
This is to create several methods to create different boundary panels, like size, background colour for the outline, and the location.

Button.java



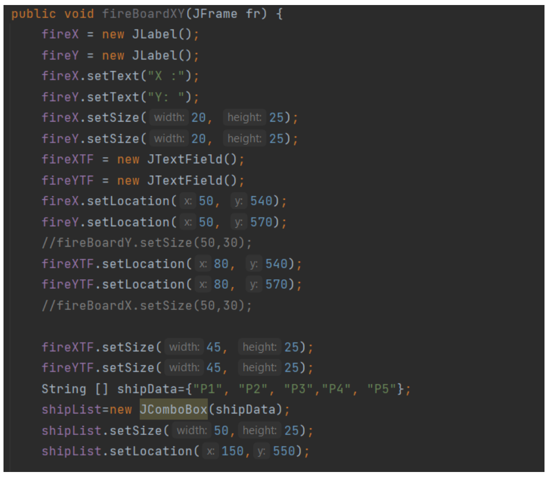
```
public class Button {
    // JButton objects representing various buttons in the GUI.
    6 usages
    JButton buttonStart, buttonFire,buttonStop,buttonPause, buttonReplay,buttonConfirm;

    // JLabel objects representing labels for the X and Y coordinates.
    6 usages
    JLabel fireX,fireY;

    // JTextField objects representing text fields for entering X and Y coordinates.
    5 usages
    JTextField fireXTF,fireYTF;

    // JComboBox object representing a drop-down list for selecting a ship.
    4 usages
    JComboBox shipList;
```

This method is to declare the instance variables for buttons in GUI.



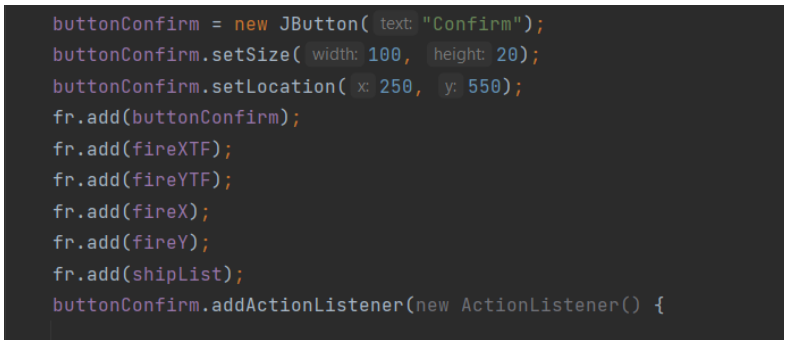
```java
public void putButton(JFrame fr) {
    // Sets their properties (size, location, text, and background color), and adds them to the frame.
    buttonStart = new JButton();
    buttonFire = new JButton();
    buttonPause = new JButton();
    buttonReplay = new JButton();
    buttonStart.setSize( width: 80,  height: 30);
    buttonStart.setLocation( x: 400,  y: 560);
    buttonStart.setText("START");
    buttonStart.setBackground(Color.ORANGE);
    buttonFire.setSize( width: 80,  height: 30);
    buttonFire.setLocation( x: 280,  y: 560);
    buttonFire.setText("FIRE");
    buttonFire.setBackground(Color.red);
    buttonPause.setText("PAUSE");
    buttonPause.setSize( width: 80,  height: 30);
    buttonPause.setLocation( x: 520,  y: 560);
    buttonPause.setBackground(Color.ORANGE);
    buttonStop = new JButton();
    buttonStop.setSize( width: 80,  height: 30);
    buttonStop.setLocation( x: 400,  y: 560);
    buttonStop.setText("STOP");
    buttonStop.setBackground(Color.ORANGE);
    buttonReplay.setText("REPLAY");
    buttonReplay.setSize( width: 80,  height: 30);
    buttonReplay.setLocation( x: 520,  y: 560);
    buttonReplay.setBackground(Color.ORANGE);
    fr.add(buttonStart);
    fr.add(buttonFire);
    fr.add(buttonPause);
```

This is responsible for creating and configuring buttons and adding them to the provided JFrame object (fr).



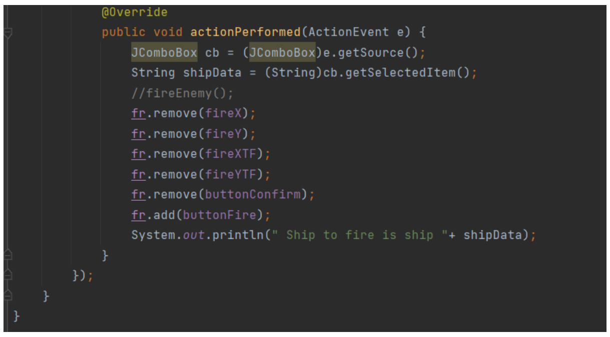
```java
public void fireBoardXY(JFrame fr) {
    fireX = new JLabel();
    fireY = new JLabel();
    fireX.setText("X :");
    fireY.setText("Y: ");
    fireX.setSize( width: 20,  height: 25);
    fireY.setSize( width: 20,  height: 25);
    fireXTF = new JTextField();
    fireYTF = new JTextField();
    fireX.setLocation( x: 50,  y: 540);
    fireY.setLocation( x: 50,  y: 570);
    //fireBoardY.setSize(50,30);
    fireXTF.setLocation( x: 80,  y: 540);
    fireYTF.setLocation( x: 80,  y: 570);
    //fireBoardX.setSize(50,30);

    fireXTF.setSize( width: 45,  height: 25);
    fireYTF.setSize( width: 45,  height: 25);
    String [] shipData={"P1", "P2", "P3","P4", "P5"};
    shipList=new JComboBox(shipData);
    shipList.setSize( width: 50, height: 25);
    shipList.setLocation( x: 150, y: 550);
```

This is responsible for creating components related to firing coordinates (X and Y) and ship selection. Later it will be added to the frame.



```java
buttonConfirm = new JButton( text: "Confirm");
buttonConfirm.setSize( width: 100,  height: 20);
buttonConfirm.setLocation( x: 250,  y: 550);
fr.add(buttonConfirm);
fr.add(fireXTF);
fr.add(fireYTF);
fr.add(fireX);
fr.add(fireY);
fr.add(shipList);
buttonConfirm.addActionListener(new ActionListener() {
```

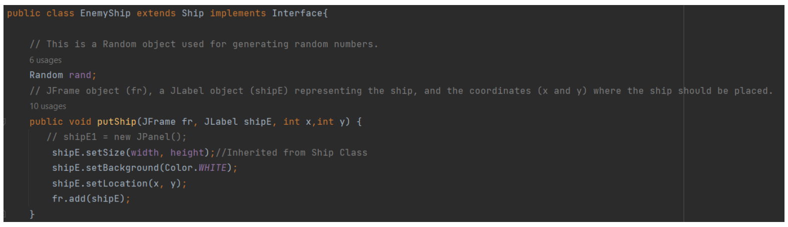
It also creates a "Confirm" button (buttonConfirm) and adds an action listener to it.



```java
            @Override
            public void actionPerformed(ActionEvent e) {
                JComboBox cb = (JComboBox)e.getSource();
                String shipData = (String)cb.getSelectedItem();
                //fireEnemy();
                fr.remove(fireX);
                fr.remove(fireY);
                fr.remove(fireXTF);
                fr.remove(fireYTF);
                fr.remove(buttonConfirm);
                fr.add(buttonFire);
                System.out.println(" Ship to fire is ship "+ shipData);
            }
        });
    }
}
```

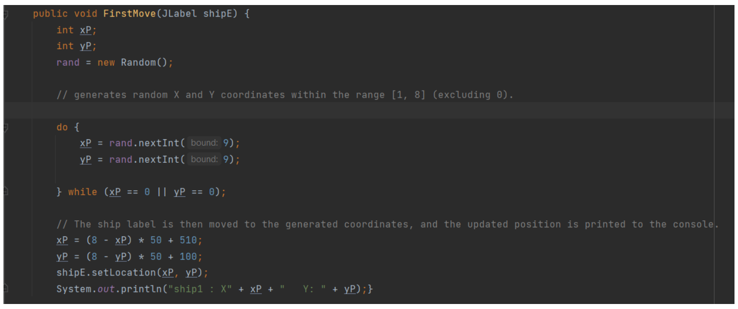
When the "Confirm" button is clicked, the selected ship data from the ship list is retrieved, and the X and Y labels, input fields, and confirm button are removed.

EnemyShip.java



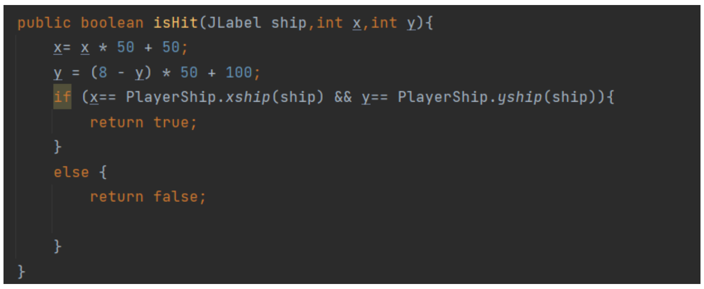
```java
public class EnemyShip extends Ship implements Interface{

    // This is a Random object used for generating random numbers.
    6 usages
    Random rand;
    // JFrame object (fr), a JLabel object (shipE) representing the ship, and the coordinates (x and y) where the ship should be placed.
    10 usages
    public void putShip(JFrame fr, JLabel shipE, int x,int y) {
      // shipE1 = new JPanel();
        shipE.setSize(width, height);//Inherited from Ship Class
        shipE.setBackground(Color.WHITE);
        shipE.setLocation(x, y);
        fr.add(shipE);
    }
```

This method represents the enemy ship in the game and is responsible for placing an enemy ship on the game board.



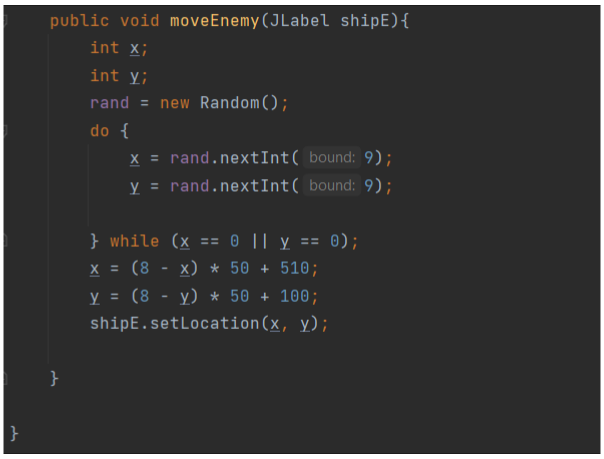
```java
    public void FirstMove(JLabel shipE) {
        int xP;
        int yP;
        rand = new Random();

        // generates random X and Y coordinates within the range [1, 8] (excluding 0).

        do {
            xP = rand.nextInt( bound: 9);
            yP = rand.nextInt( bound: 9);

        } while (xP == 0 || yP == 0);

        // The ship label is then moved to the generated coordinates, and the updated position is printed to the console.
        xP = (8 - xP) * 50 + 510;
        yP = (8 - yP) * 50 + 100;
        shipE.setLocation(xP, yP);
        System.out.println("ship1 : X" + xP + "   Y: " + yP);}
```

This is responsible for the initial movement of the enemy ship.



```
public boolean isHit(JLabel ship,int x,int y){
    x= x * 50 + 50;
    y = (8 - y) * 50 + 100;
    if (x== PlayerShip.xship(ship) && y== PlayerShip.yship(ship)){
        return true;
    }
    else {
        return false;

    }
}
```

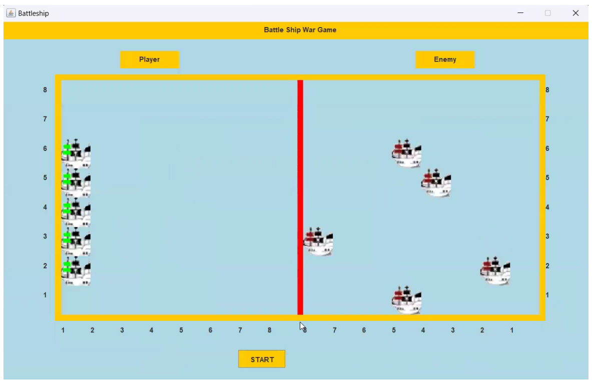
This checks if the player's shot hit the enemy ship.



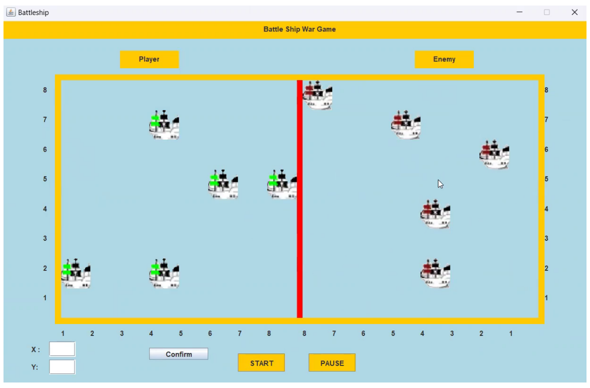
```
    public void moveEnemy(JLabel shipE){
        int x;
        int y;
        rand = new Random();
        do {
            x = rand.nextInt( bound: 9);
            y = rand.nextInt( bound: 9);

        } while (x == 0 || y == 0);
        x = (8 - x) * 50 + 510;
        y = (8 - y) * 50 + 100;
        shipE.setLocation(x, y);

    }

}
```

This is responsible for moving the enemy ship to a new random position.

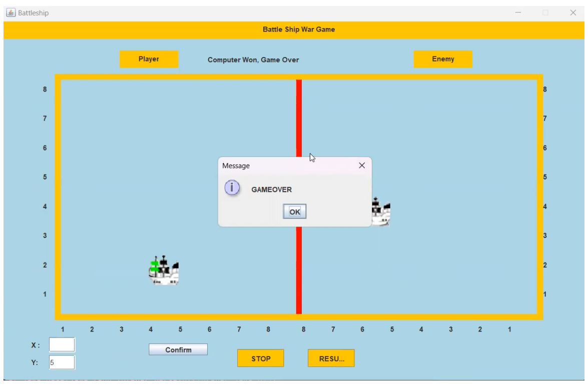
IV. Evidence of Working Program



Evidence where the game has not started yet. This is the main menu when you run the code.



Evidence where the game is played. This is when you pressed the fire button.



Computer Won, Game Over

Message

GAMEOVER

OK

Evidence where the player either wins or loses the game, the GUI shows the bar “Game Over!”.

V. Sources

- https://www.javatpoint.com/java-swing
- https://www.javatpoint.com/java-awt
- https://docs.oracle.com/javase/8/docs/api/java/util/package-summary.html
- https://youtu.be/vO7wHV0HB8w
- https://youtu.be/BaLBZEwchQY