Bubble Destroyer

Presented By

Kanokprapha Kawilo 6604062661016

Presented to

Asst.Prof.Sathit Prasomphan

Object-Oriented Programming 040613204 Semister 1 / 2024

Department of Computer and Information Science, Faculty of Applied Science

KING MONGKUT'S UNIVERSITY OF TECHNOLOGY NORTH BANGKOK

Chapter 1

Introduction

1.1 Background and Significance

This project was created to assess learning outcomes in the course "Object-Oriented Programming" (OOP) by applying the content from lessons to build a game. The game is designed for single-player mode to enhance concentration, develop quick thinking, and provide stress relief.

1.2 Objectives

1.2.1 To apply the knowledge gained from the course in a practical project

1.2.2 To gain hands-on experience in Java programming

1.3 Scope of the Project

1.3.1 Java is used as the primary programming language

1.3.2 The project is designed and developed following Object-Oriented Programming (OOP) principles

1.4 Work Plan Schedule

Order	List	9 – 13 OCT	14 OCT –3 NOV	4 -6 NOV	%
1	Design game				10 %
2	Study related information				20 %
3	Write code				60 %
4	Add code details				65 %
5	Edit code				80 %
6	Test code				85 %
7	compile a document				90 %
8	Check for errors				100 %

Chapter 2

Game

2.1 Details

A dangerous situation has arisen in space as many rocks and meteor fragments are floating towards us. Little Bubble has been given the important task of protecting space with the ability to destroy all dangerous objects. To prevent these threats from causing harm, Little Bubble serves as the guardian, ensuring the safety of the space we inhabit.

2.2 Game Details

A dangerous situation has arisen in space as many rocks and meteor fragments are floating towards us. Little Bubble has been given the important task of protecting space with the ability to destroy all dangerous objects. To prevent these threats from causing harm, Little Bubble serves as the guardian, ensuring the safety of the space we inhabit.

2.3 How to play

Press control movement.

- Press w to move up.

- Press s to move down.

- Press a to move left.

- Press d to move right.

- Press f to rotate left.

- Press r to rotate right.

- Press enter to shoot.

2.4 Storyboard

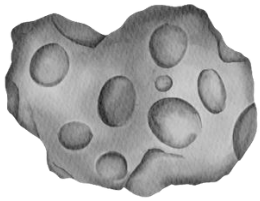
Character

Bubble

player

Destroyter

score + 1.



Rock

score + 2.

Bom

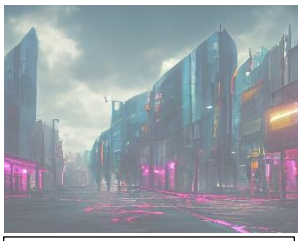
Total score - 5.

Blood

health 25%

Page Menu

Level 1

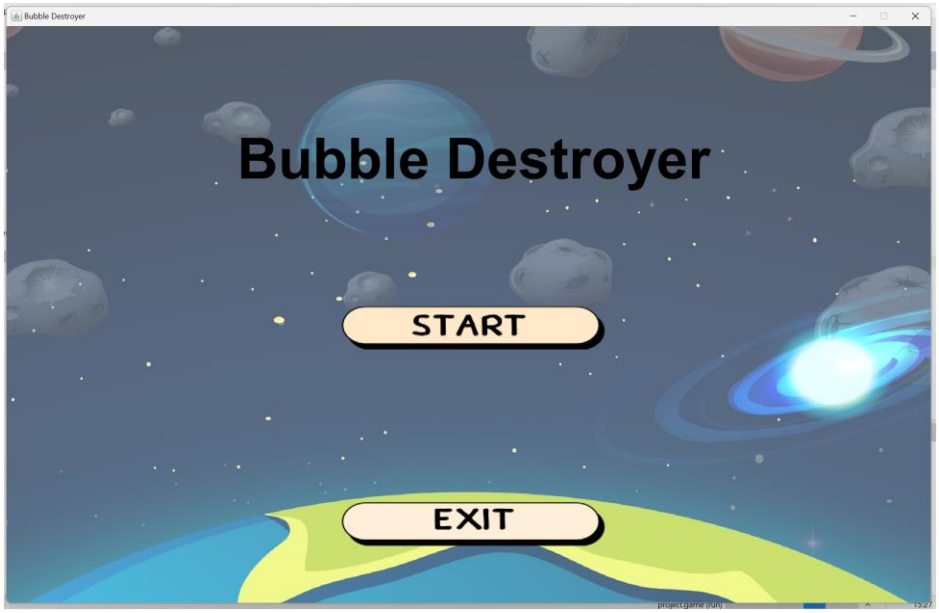


Level 2

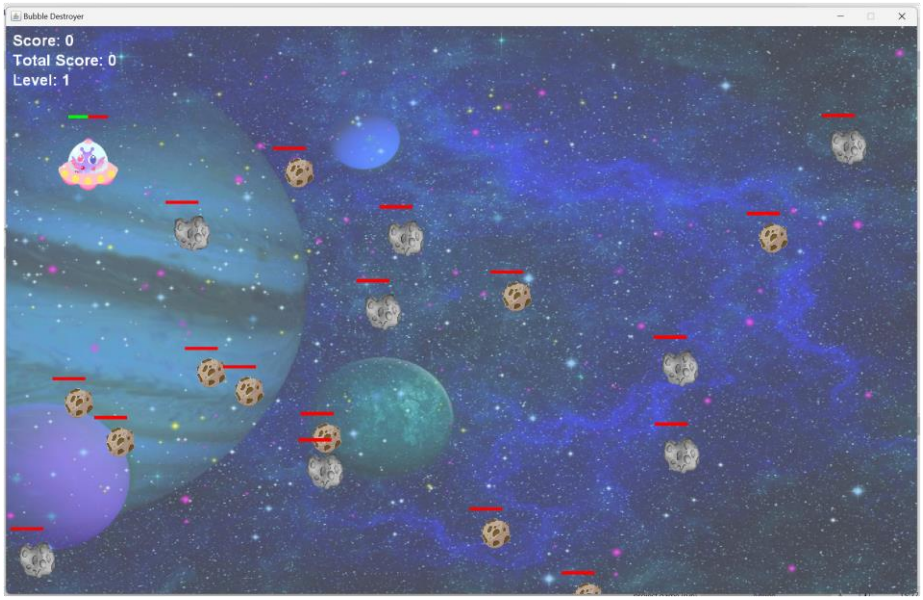
Level 3

Scene

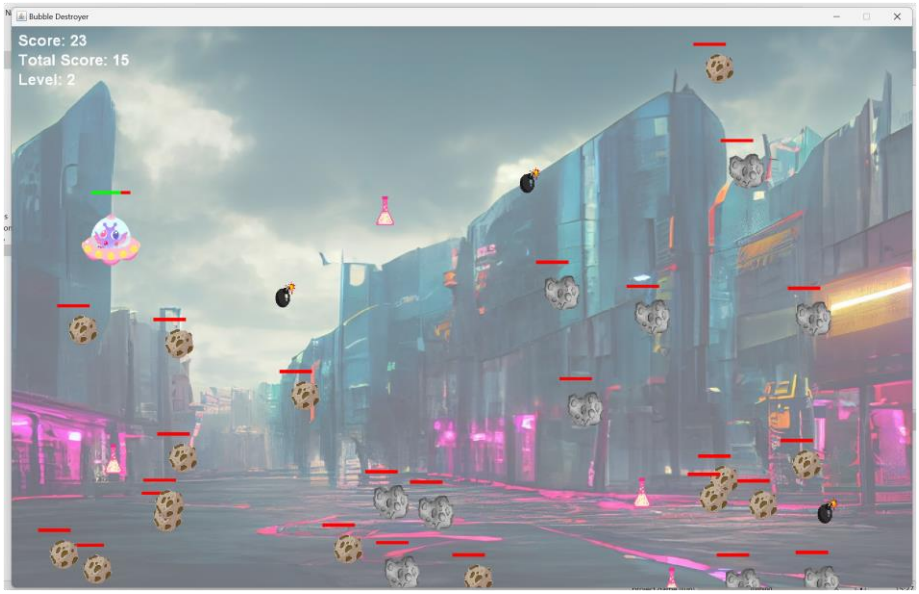
Press Start to begin the game. Press Exit to exit the game.



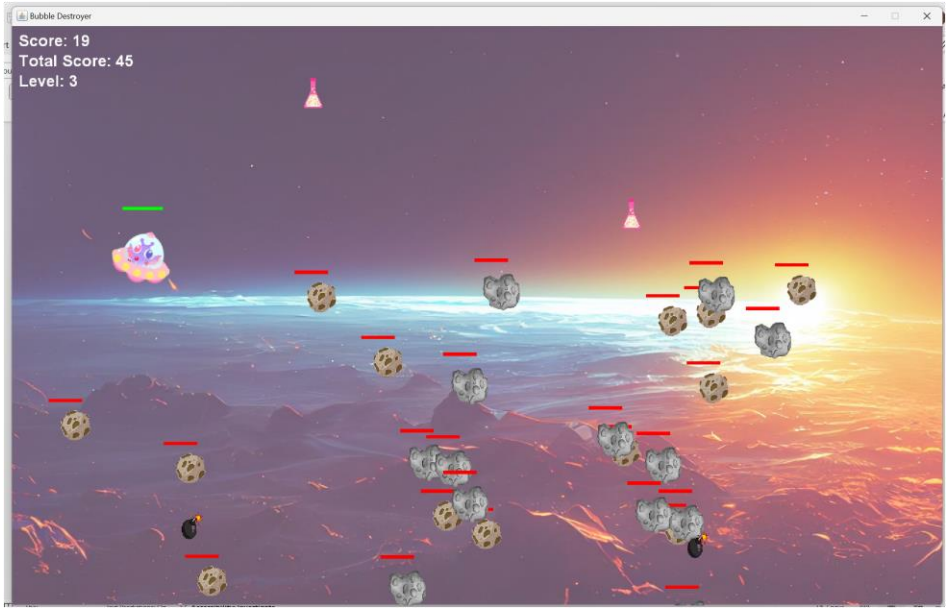
When you press Start, the game will begin at Level 1. You need to score 20 points to advance to the next level.



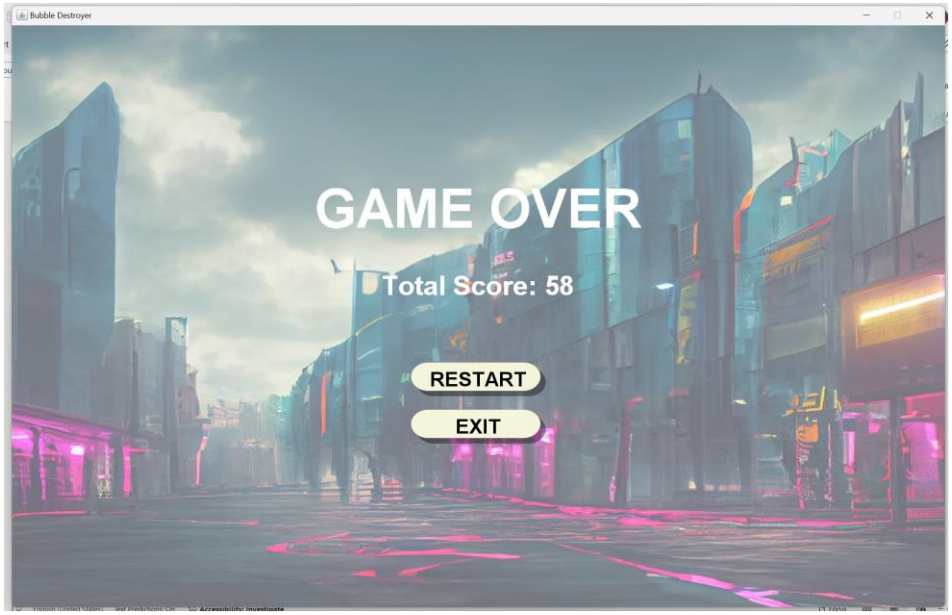
When you reach 20 points, the game will advance to Level 2. The score from Level 1 will be added to the total score, and additional bombs and health will be provided. In Level 2, you need to score 40 points to move on to the next level.



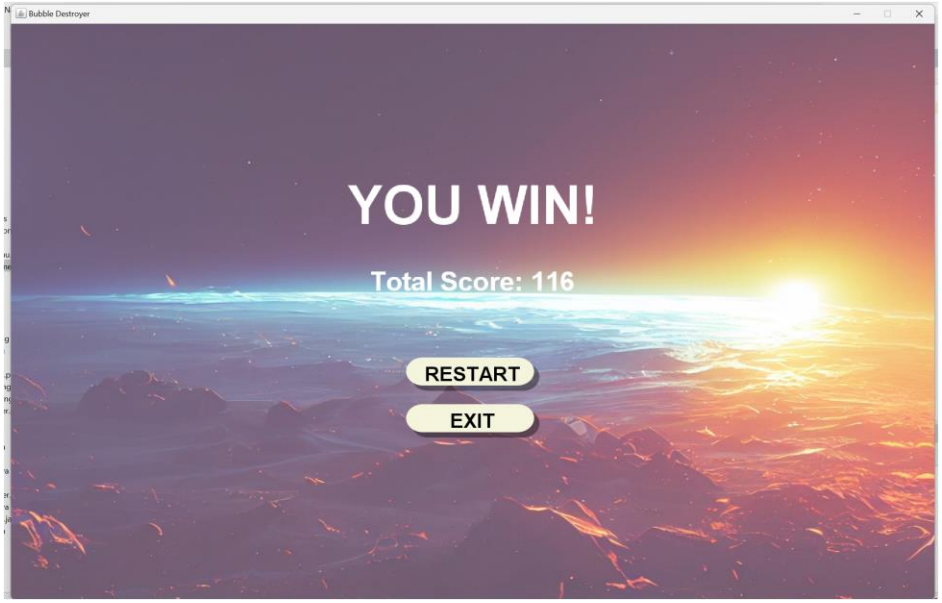
When you reach 40 points, the game will advance to Level 3. The scores from Levels 1 and 2 will be added to the total score, with additional bombs and health provided. In this level, you must eliminate all Destroyers and Rocks to win.



Page Game Over!!! Background will Level.

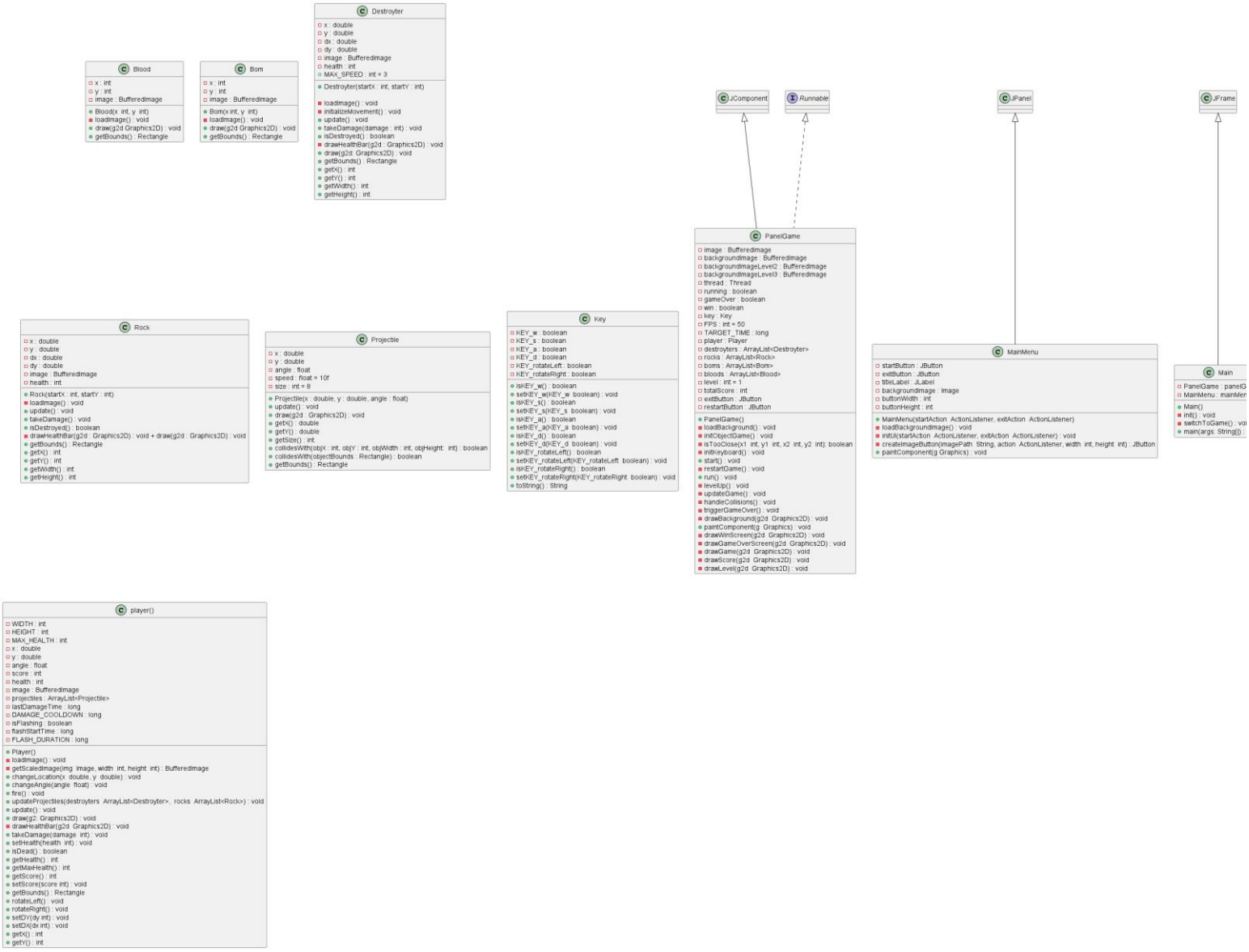


Page YOU WIN!.

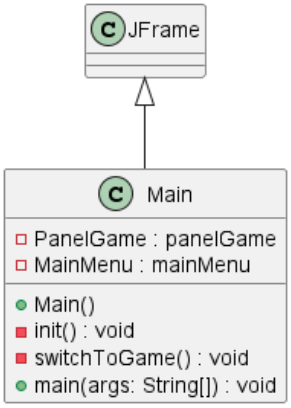


2.5 Class Diagram

There are a total of 10 class diagrams.



1.Class Main



2.Class PanelGame

3.Class Player

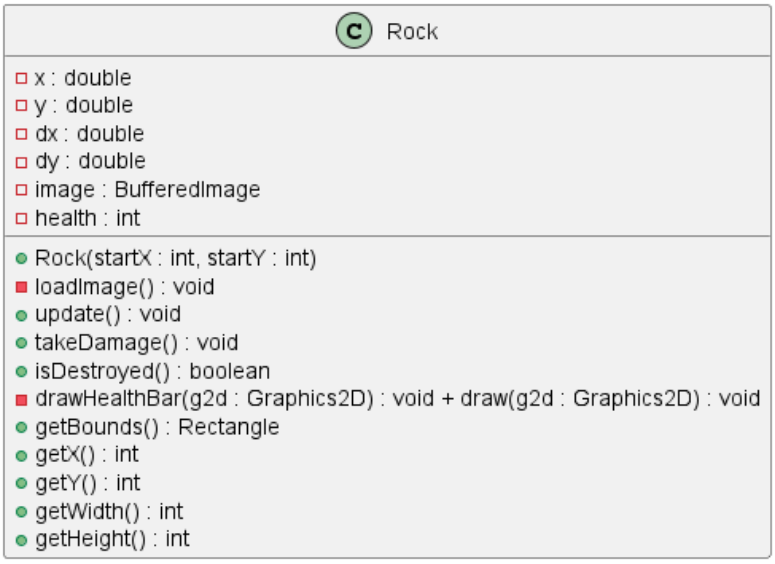
```
player()
- WIDTH : int
- HEIGHT : int
- MAX_HEALTH : int
- x : double
- y : double
- angle : float
- score : int
- health : int
- image : BufferedImage
- projectiles : ArrayList<Projectile>
- lastDamageTime : long
- DAMAGE_COOLDOWN : long
- isFlashing : boolean
- flashStartTime : long
- FLASH_DURATION : long
+ Player()
# loadImage() : void
# getScaledImage(img  Image, width  int, height  int) : BufferedImage
+ changeLocation(x  double, y  double) : void
+ changeAngle(angle  float) : void
+ fire() : void
+ updateProjectiles(destroyters  ArrayList<Destroyter>,  rocks  ArrayList<Rock>) : void
+ update() : void
+ draw(g2: Graphics2D) : void
# drawHealthBar(g2d  Graphics2D) : void
+ takeDamage(damage  int) : void
+ setHealth(health  int) : void
+ isDead() : boolean
+ getHealth() : int
+ getMaxHealth() : int
+ getScore() : int
+ setScore(score int) : void
+ getBounds() : Rectangle
+ rotateLeft() : void
+ rotateRight() : void
+ setDY(dy int) : void
+ setDX(dx int) : void
+ getX() : int
+ getY() : int
```

4.Class Destroyter

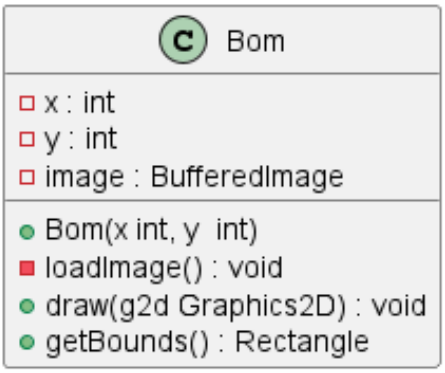
```
Destroyter
- x : double
- y : double
- dx : double
- dy : double
- image : BufferedImage
- health : int
- MAX_SPEED : int = 3
+ Destroyter(startX : int, startY : int)

# loadImage() : void
# initializeMovement() : void
+ update() : void
+ takeDamage(damage : int) : void
+ isDestroyed() : boolean
# drawHealthBar(g2d : Graphics2D) : void
+ draw(g2d: Graphics2D) : void
+ getBounds() : Rectangle
+ getX() : int
+ getY() : int
+ getWidth() : int
+ getHeight() : int
```

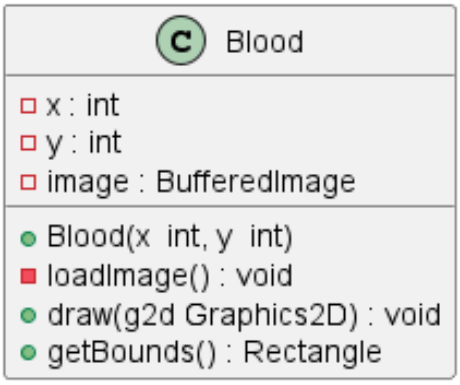
5.Class Rock



6.Class Bom



7.Class Blood



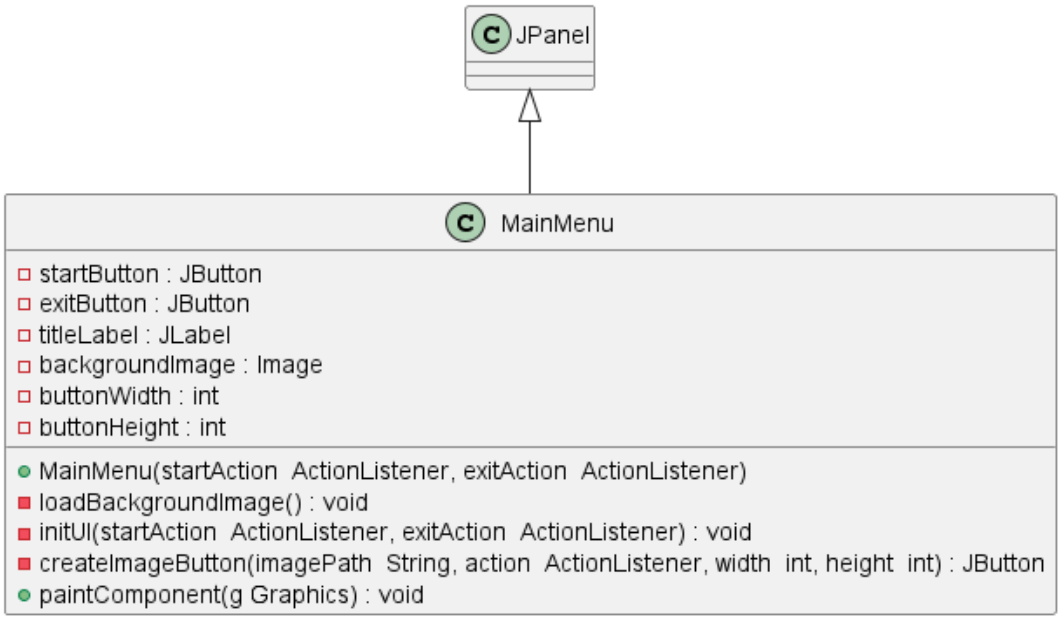
8.Class Projectile

```
Projectile
- x : double
- y : double
- angle : float
- speed : float = 10f
- size : int = 8
+ Projectile(x : double, y : double, angle : float)
+ update() : void
+ draw(g2d : Graphics2D) : void
+ getX() : double
+ getY() : double
+ getSize() : int
+ collidesWith(objX : int, objY : int, objWidth : int, objHeight:  int) : boolean
+ collidesWith(objectBounds : Rectangle) : boolean
+ getBounds() : Rectangle
```

9.Class Key

```
Key
- KEY_w : boolean
- KEY_s : boolean
- KEY_a : boolean
- KEY_d : boolean
- KEY_rotateLeft : boolean
- KEY_rotateRight : boolean
+ isKEY_w() : boolean
+ setKEY_w(KEY_w  boolean) : void
+ isKEY_s() : boolean
+ setKEY_s(KEY_s  boolean) : void
+ isKEY_a() : boolean
+ setKEY_a(KEY_a  boolean) : void
+ isKEY_d() : boolean
+ setKEY_d(KEY_d  boolean) : void
+ isKEY_rotateLeft() : boolean
+ setKEY_rotateLeft(KEY_rotateLeft  boolean) : void
+ isKEY_rotateRight() : boolean
+ setKEY_rotateRight(KEY_rotateRight  boolean) : void
+ toString() : String
```

10.Class MainMenu



Code

1.Class Main

```java
public class Main extends JFrame {

    private PanelGame panelGame;
    private MainMenu mainMenu;

    public Main() {
        init();
    }
```

Constructor must implement the init() method in addition to the methods used to initialize the game window by the init() method.

Encapsulation hiding the inner workings of variables and methods related to panelGame and mainMenu that are set to private

```java
    private void init() {
        setTitle("Bubble Destroyer");
        setSize(1400, 900);
        setLocationRelativeTo(null);
        setResizable(false);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        panelGame = new PanelGame();
        mainMenu = new MainMenu (new StartGameAction(), e -> System.exit(0));

        add(mainMenu, BorderLayout.CENTER);

        addWindowListener(new WindowAdapter() {
            @Override
            public void windowOpened(WindowEvent e) {
                panelGame.start();
            }
        });
    }
```

Composition
PanelGame and MainMenu as components for creating the game window.
In the switchToGame() method there is a switch from mainMenu to panelGame.

`private class StartGameAction implements ActionListener {`
`  @Override`
`  public void actionPerformed(ActionEvent e) {`
`  switchToGame();  // Switch to the game when "Start Game" is clicked`
`  }`
`}`
	Event Handling is used for button clicks and window openings: StartGameAction Used to handle the "Start Game" button click event to switch to the game. addWindowListener Used to handle the windowOpened event to start the game. (panelGame.start()) when the window opens.

`private class StartGameAction implements ActionListener {`
`  @Override`
`  public void actionPerformed(ActionEvent e) {`
`  switchToGame();  // Switch to the game when "Start Game" is clicked`
`  }`
`}`
	Polymorphism is used in the StartGameAction section which implements the ActionListener interface. and use it through the method actionPerformed

2.Class PanelGame

`public PanelGame() {`
`  this.setPreferredSize(new Dimension(1300, 900));`
`  setFocusable(true);`
`  requestFocusInWindow();`
`  initObjectGame();`
`  initKeyboard();`
`  loadBackground();`
`  start();`
	Constructor is used to initialize the game's settings, such as screen size, background image loading. and UI settings initObjectGame() method is called to create game objects such as Player, Destroyter, Rock, Bom, and Blood, and initKeyboard() is to set the keyboard.

```java
public class PanelGame extends JComponent implements Runnable {
    private BufferedImage image;
    private BufferedImage backgroundImage;
    private BufferedImage backgroundImageLevel2;
    private BufferedImage backgroundImageLevel3;
    private Thread thread;
    private boolean running = false;
    private boolean gameOver = false;
    private boolean win = false; // Flag to indicate if the player has won
    private Key key;
    private final int FPS = 50;
    private final long TARGET_TIME = 1000000000 / FPS;

    private Player player;
    private ArrayList<Destroyter> destroyters;
    private ArrayList<Rock> rocks;
    private ArrayList<Bom> boms;
    private ArrayList<Blood> bloods;
    private static final int NUM_OBSTACLES = 10;
    private int level = 1;
    private int totalScore = 0;

    // Minimum safe distance for object spawn
    private static final int MIN_SAFE_DISTANCE = 100;
    private JButton exitButton;
    private JButton restartButton;
```

```java
    private void updateGame() {
        // Update player movement based on key states
        player.setDY(0);
        player.setDX(0);
        if (key.isKEY_w()) player.setDY(-2);
        if (key.isKEY_s()) player.setDY(2);
        if (key.isKEY_a()) player.setDX(-2);
        if (key.isKEY_d()) player.setDX(2);
        if (key.isKEY_rotateLeft()) player.rotateLeft();
        if (key.isKEY_rotateRight()) player.rotateRight();

        player.update();
        player.updateProjectiles(destroyters, rocks);

        handleCollisions();
        // Check win condition in level 3
        if (level == 3 && destroyters.isEmpty() && rocks.isEmpty()) {
            totalScore += player.getScore(); // Add level 3 score to totalScore
            win = true;
            running = false; // Stop the game loop
            return;
        }
        // Level up conditions
        if ((level == 1 && player.getScore() >= 20) || (level == 2 && player.getScore() >= 40))
            levelUp();
    }
}
```

Composition

PanelGame combines the Player, Destroyter, Rock, Bom, and Blood objects to create various game elements in initObjectGame().

Encapsulation in PanelGame prevents access to private variables such as players, destroyters, rocks, boms, bloods, which are only used within the class. These are accessed and manipulated only through methods within the class These are accessed and manipulated only through methods within the class (e.g., updateGame(), handleCollisions()).

Polymorphism occurs when updating and drawing in-game objects, such as Destroyter, Rock, Bom, and Blood, which are stored in their own separate ArrayList, and each object has a different behavior in the draw and update

```java
private void handleCollisions() {
    Iterator<Destroyter> destroyterIterator = destroyters.iterator();
    while (destroyterIterator.hasNext()) {
        Destroyter destroyter = destroyterIterator.next();
        destroyter.update();
        if (destroyter.getBounds().intersects(player.getBounds())) {
            player.takeDamage(25); // Reduce health by 25
            destroyterIterator.remove();
            if (player.getHealth() <= 0) {
                triggerGameOver();
                return;
            }
        }
    }

    Iterator<Rock> rockIterator = rocks.iterator();
    while (rockIterator.hasNext()) {
        Rock rock = rockIterator.next();
        rock.update();
        if (rock.getBounds().intersects(player.getBounds())) {
            player.takeDamage(25);
            rockIterator.remove();
            if (player.getHealth() <= 0) {
                triggerGameOver();
                return;
            }
        }
    }
    Iterator<Bom> bomIterator = boms.iterator();
    while (bomIterator.hasNext()) {
        Bom bom = bomIterator.next();
        if (bom.getBounds().intersects(player.getBounds())) {
            totalScore = Math.max(0, totalScore - 5); // Ensure score does not go below zero
            bomIterator.remove();
        }
    }

    Iterator<Blood> bloodIterator = bloods.iterator();
    while (bloodIterator.hasNext()) {
        Blood blood = bloodIterator.next();
        if (blood.getBounds().intersects(player.getBounds())) {
            player.setHealth(Math.min(player.getHealth() + 25, player.getMaxHealth()));
            bloodIterator.remove();
        }
    }
}
```

methods based on its characteristics. each class

```java
private void initKeyboard() {
    key = new Key();
    addKeyListener(new KeyAdapter() {
        @Override
        public void keyPressed(KeyEvent e) {
            if (gameOver || win) return;
            switch (e.getKeyCode()) {
                case KeyEvent.VK_W -> key.setKEY_w(true);
                case KeyEvent.VK_S -> key.setKEY_s(true);
                case KeyEvent.VK_A -> key.setKEY_a(true);
                case KeyEvent.VK_D -> key.setKEY_d(true);
                case KeyEvent.VK_F -> key.setKEY_rotateLeft(true);
                case KeyEvent.VK_R -> key.setKEY_rotateRight(true);
                case KeyEvent.VK_ENTER -> player.fire();
            }
        }
        @Override
        public void keyReleased(KeyEvent e) {
            switch (e.getKeyCode()) {
                case KeyEvent.VK_W -> key.setKEY_w(false);
                case KeyEvent.VK_S -> key.setKEY_s(false);
                case KeyEvent.VK_A -> key.setKEY_a(false);
                case KeyEvent.VK_D -> key.setKEY_d(false);
                case KeyEvent.VK_F -> key.setKEY_rotateLeft(false);
                case KeyEvent.VK_R -> key.setKEY_rotateRight(false);
            }
        }
    });
```

Event Handling

This is done via the keyboard using the KeyAdapter, which captures key presses and releases to control player movement and character rotation.

```java
public void start() {
    if (thread == null || !running) {
        thread = new Thread(this);
        running = true;
        thread.start();
    }
}

@Override
public void run() {
    while (running) {
        long startTime = System.nanoTime();

        if (!gameOver && !win) {
            updateGame();
        }
        repaint();

        long elapsed = System.nanoTime() - startTime;
        long wait = (TARGET_TIME - elapsed) / 1000000;

        if (wait < 0) {
            wait = 5;
        }

        try {
            Thread.sleep(wait);
        } catch (InterruptedException e) {
            System.err.println(e);
        }
    }
}
```

Thread for continuous game status updates. Using the run() method

3.Class Player

```java
public Player() {
    loadImage();
    projectiles = new ArrayList<>();
}
```

Constructor is used for basic initialization, such as loading the player image and creating an ArrayList for projectiles.

```java
public class Player {
    public static final int WIDTH = 150;
    public static final int HEIGHT = 120;
    public static final int MAX_HEALTH = 100;

    private double x;
    private double y;
    private float angle = 0f;
    private int score = 0;
    private int health = MAX_HEALTH;
```

Encapsulation Prevent direct access to important variables. By setting variables like health, score, and projectiles to private, and creating getters

```java
// Getters for health, max health, and score
public int getHealth() {
    return health;
}

public int getMaxHealth() {
    return MAX_HEALTH;
}

public int getScore() {
    return score;
}

// Setter for score to reset or adjust the player's score
public void setScore(int score) {
    this.score = score;
}
```

and setters to manage external access to these values.

```java
// Update projectiles, checking for collisions with destroyers and rocks
public void updateProjectiles(ArrayList<Destroyter> destroyters, ArrayList<Rock> rocks) {
    synchronized (projectiles) {
        Iterator<Projectile> projectileIterator = projectiles.iterator();
        while (projectileIterator.hasNext()) {
            Projectile projectile = projectileIterator.next();
            projectile.update();

            boolean shouldRemoveProjectile = false;

            // Remove projectiles that go outside screen bounds
            if (projectile.getX() < 0 || projectile.getX() > 1400 || projectile.getY() < 0 || projectile.getY() > 900) {
                projectileIterator.remove();
                continue;
            }

            // Check collision with destroyers
            Iterator<Destroyter> destroyterIterator = destroyters.iterator();
            while (destroyterIterator.hasNext()) {
                Destroyter destroyter = destroyterIterator.next();
                if (projectile.collidesWith(destroyter.getBounds())) {
                    score += 1;
                    destroyterIterator.remove(); // Remove the Destroyter
                    shouldRemoveProjectile = true;
                    break;
                }
            }

            // Check collision with rocks if no collision with destroyers
            if (!shouldRemoveProjectile) {
                Iterator<Rock> rockIterator = rocks.iterator();
                while (rockIterator.hasNext()) {
                    Rock rock = rockIterator.next();
                    if (projectile.collidesWith(rock.getBounds())) {
                        score += 2;
                        rockIterator.remove(); // Remove the Rock
                        shouldRemoveProjectile = true;
                        break;
                    }
                }
            }

            if (shouldRemoveProjectile) {
                projectileIterator.remove();
            }
        }
    }
}
```

Polymorphism occurs when manipulating projectiles fired by players in a method. updateProjectiles() which calls the update() method for each projectile in ArrayList<Projectile> projectiles

4.Class Destroyter

`public Destroyter(int startX, int startY) {`
`  this.x = startX;`
`  this.y = startY;`
`  loadImage();`
`  initializeMovement();`
`}`
	Constructor of the Destroyter class is public Destroyter

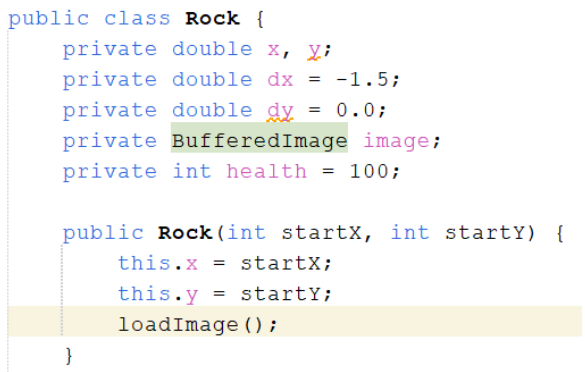
`public class Destroyter {`
`  private double x, y;`
`  private double dx;`
`  private double dy = 0;`
`  private BufferedImage image;`
`  private int health = 100;`
`  private static final int MAX_SPEED = 3;`
 	
`  // Getters`
`  public int getX() { return (int)x; }`
`  public int getY() { return (int)y; }`
`  public int getWidth() { return image != null ? image.getWidth() : 80; }`
`  public int getHeight() { return image != null ? image.getHeight() : 60; }`
`}`
	Encapsulation By setting important variables such as x, y, dx, dy, and health as private to prevent direct access from outside the class. These variables are accessed through public methods.

`private void loadImage() {`
`  try {`
`  image = ImageIO.read(getClass().getResource("/game/image/destroyter.png"));`
`  } catch (IOException e) {`
`  System.err.println("Error loading destroyter image: " + e.getMessage());`
`  }`
`}`
	Composition in this class is represented by a combination. BufferedImage for the Destroyter's image and use Random for the default enemy movement setting dx by BufferedImage It is loaded via the loadImage() method.

5.Class Rock

`public Rock(int startX, int startY) {`
`  this.x = startX;`
`  this.y = startY;`
`  loadImage();`
`}`
	Constructor public Rock(int startX, int startY) is used to set the starting position of the Rock object and call the loadImage() method to load the rock image:

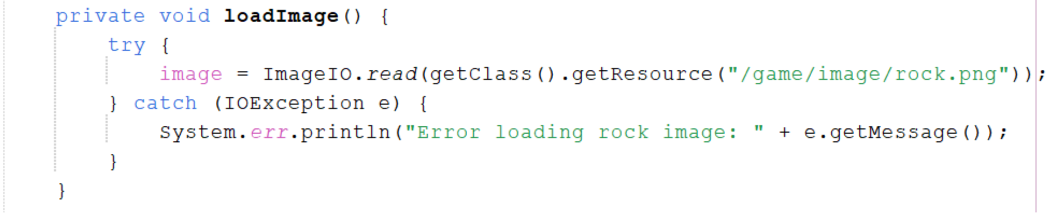
	Encapsulation By setting the x, y, dx, dy, and health variables to private, this prevents direct access from



```java
public class Rock {
    private double x, y;
    private double dx = -1.5;
    private double dy = 0.0;
    private BufferedImage image;
    private int health = 100;

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

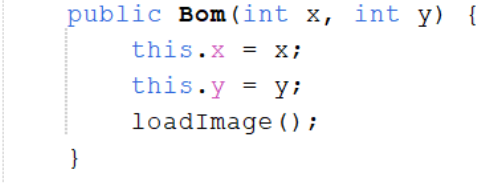
outside the class. and allows access only through the public method.



```java
    private void loadImage() {
        try {
            image = ImageIO.read(getClass().getResource("/game/image/rock.png"));
        } catch (IOException e) {
            System.err.println("Error loading rock image: " + e.getMessage());
        }
    }
```

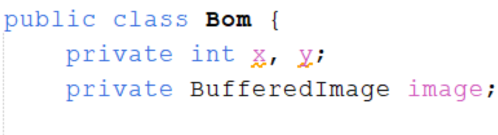
Composition is used in Rock by combining a BufferedImage. for pictures of rocks It is loaded by the loadImage() method within the class.

6.Class Bom



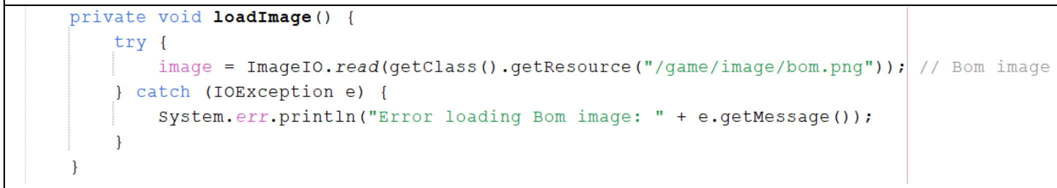
```java
    public Bom(int x, int y) {
        this.x = x;
        this.y = y;
        loadImage();
    }
```

Constructor is public Bom(int x, int y) used to set the initial position (x, y) of the explosive (Bom) and call the loadImage() method to load the image to be displayed.



```java
public class Bom {
    private int x, y;
    private BufferedImage image;
```

Encapsulation By setting important variables such as x, y, and image as private to prevent direct access from outside the class.



```java
    private void loadImage() {
        try {
            image = ImageIO.read(getClass().getResource("/game/image/bom.png")); // Bom image
        } catch (IOException e) {
            System.err.println("Error loading Bom image: " + e.getMessage());
        }
    }
```

Composition is used in Bom class with BufferedImage. is a component which is used to store images of Bom by BufferedImage It is loaded via the loadImage() method.

7.Class Blood

`public Blood(int x, int y) {`
`  this.x = x;`
`  this.y = y;`
`  loadImage();`
`}`
	Constructor is public Blood(int x, int y) used to set the initial position (x, y) of the Blood object and call the loadImage() method to load the image to be displayed.

`public class Blood {`
`  private int x, y;`
`  private BufferedImage image;`
	Encapsulation By setting important variables such as x, y and image as private to prevent direct outside access.

`private void loadImage() {`
`  try {`
`  image = ImageIO.read(getClass().getResource("/game/image/blood.png"));`
`  if (image == null) {`
`  throw new IOException("Blood image not found at specified path.");`
`  }`
`  } catch (IOException e) {`
`  System.err.println("Error loading Blood image: " + e.getMessage());`
`  }`
`}`
	Composition is used in the Blood class using a BufferedImage. For storing images of Blood which are loaded via the loadImage() method.

8.Class Projectile

`public Projectile(double x, double y, float angle) {`
`  this.x = x;`
`  this.y = y;`
`  this.angle = angle;`
`}`
	Constructor of the Projectile class is public Projectile(double x, double y, float angle) and is used to set the initial position (x, y) and angle (angle) of the projectile.

```java
public class Projectile {
    private double x;
    private double y;
    private final float angle;
    private final float speed = 10f;
    private final int size = 8;

    public Projectile(double x, double y, float angle) {
        this.x = x;
        this.y = y;
        this.angle = angle;
    }
```

Encapsulation By making the variables x, y, angle, speed, and size private (or final for constants), this prevents direct outside access and modification.

```java
    // Helper method to get the bounds of the projectile
    public Rectangle getBounds() {
        return new Rectangle((int) x, (int) y, size, size);
    }
}
```

Composition via creating a Rectangle representing the bounds of the bullet in the getBounds() and collidesWith() methods.

```java
    // Original method: Checks collision with another object using individual coordinates and dimensions
    public boolean collidesWith(int objX, int objY, int objWidth, int objHeight) {
        Rectangle projectileBounds = new Rectangle((int) x, (int) y, size, size);
        Rectangle objectBounds = new Rectangle(objX, objY, objWidth, objHeight);
        return projectileBounds.intersects(objectBounds);
    }

    // Overloaded method: Checks collision with another object using a Rectangle
    public boolean collidesWith(Rectangle objectBounds) {
        Rectangle projectileBounds = new Rectangle((int) x, (int) y, size, size);
        return projectileBounds.intersects(objectBounds);
    }
```

Polymorphism is implemented in the form of overloading the collidesWith() method. The parameters are coordinates (objX, objY) and dimensions (objWidth, objHeight).
Rectangle to define the boundaries of objects for collision detection.

9.Class Key

```java
public class Key {
    private boolean KEY_w;
    private boolean KEY_s;
    private boolean KEY_a;
    private boolean KEY_d;
    private boolean KEY_rotateLeft;
    private boolean KEY_rotateRight;
```

Encapsulation is used explicitly in this class. By setting the variables KEY_w, KEY_s, KEY_a, KEY_d, KEY_rotateLeft, and KEY_rotateRight. and

```java
    // Getter and Setter for KEY_w
    public boolean isKEY_w() {
        return KEY_w;
    }

    public void setKEY_w(boolean KEY_w) {
        this.KEY_w = KEY_w;
    }

    // Getter and Setter for KEY_s
    public boolean isKEY_s() {
        return KEY_s;
    }

    public void setKEY_s(boolean KEY_s) {
        this.KEY_s = KEY_s;
    }

    // Getter and Setter for KEY_a
    public boolean isKEY_a() {
        return KEY_a;
    }

    public void setKEY_a(boolean KEY_a) {
        this.KEY_a = KEY_a;
    }

    // Getter and Setter for KEY_d
    public boolean isKEY_d() {
        return KEY_d;
    }

    public void setKEY_d(boolean KEY_d) {
        this.KEY_d = KEY_d;
    }

    // Getter and Setter for KEY_rotateLeft
    public boolean isKEY_rotateLeft() {
        return KEY_rotateLeft;
    }

    public void setKEY_rotateLeft(boolean KEY_rotateLeft) {
        this.KEY_rotateLeft = KEY_rotateLeft;
    }

    // Getter and Setter for KEY_rotateRight
    public boolean isKEY_rotateRight() {
        return KEY_rotateRight;
    }

    public void setKEY_rotateRight(boolean KEY_rotateRight) {
        this.KEY_rotateRight = KEY_rotateRight;
    }
```

Access to these variables is done through public getter and setter.

```java
    @Override
    public String toString() {
        return "Key{" +
                "KEY_w=" + KEY_w +
                ", KEY_s=" + KEY_s +
                ", KEY_a=" + KEY_a +
                ", KEY_d=" + KEY_d +
                ", KEY_rotateLeft=" + KEY_rotateLeft +
                ", KEY_rotateRight=" + KEY_rotateRight +
                '}';
    }
}
```

Polymorphis toString() method is overridden from the Java Object class to display the state of all variables in text.

10.Class MainMenu

```java
    public MainMenu(ActionListener startAction, ActionListener exitAction) {
        loadBackgroundImage();
        initUI(startAction, exitAction);
    }
```

Constructor of the MainMenu class is public MainMenu(ActionListener startAction, ActionListener exitAction) used to create the game's main menu. By calling the method loadBackgroundImage() to load the background image and initUI(startAction, exitAction)

```java
public class MainMenu extends JPanel {

    private JButton startButton;
    private JButton exitButton;
    private JLabel titleLabel;
    private Image backgroundImage;
    private final int buttonWidth = 450; // Adjusted button width
    private final int buttonHeight = 280; // Adjusted button height
```

Encapsulation By setting important variables such as startButton, exitButton, titleLabel, and backgroundImage to private to prevent direct access from outside the class.

```java
private void initUI(ActionListener startAction, ActionListener exitAction) {
    setLayout(null); // Use null layout for custom positioning

    // Add title label
    titleLabel = new JLabel("Bubble Destroyer");
    titleLabel.setFont(new Font("Arial", Font.BOLD, 85)); // Increased font size to 60
    titleLabel.setForeground(Color.BLACK);
    titleLabel.setHorizontalAlignment(JLabel.CENTER);
    titleLabel.setBounds(300, 150, 800, 100); // Moved down and increased width

    // Create buttons with images
    startButton = createImageButton("/game/image/START.png", startAction, buttonWidth, buttonHeight);
    exitButton = createImageButton("/game/image/EXIT.png", exitAction, buttonWidth, buttonHeight);

    add(titleLabel);
    add(startButton);
    add(exitButton);
}
```

```java
private JButton createImageButton(String imagePath, ActionListener action, int width, int height) {
    JButton button = new JButton();
    button.addActionListener(action);
    button.setFocusPainted(false);
    button.setBorderPainted(false);
    button.setContentAreaFilled(false);

    // Load and set the button image with scaling to fit the new button size
    try (InputStream imageStream = getClass().getResourceAsStream(imagePath)) {
        if (imageStream != null) {
            Image buttonImage = ImageIO.read(imageStream).getScaledInstance(width, height, Image.SCALE_SMOOTH);
            button.setIcon(new ImageIcon(buttonImage));
        } else {
            System.err.println("Button image not found at path: " + imagePath);
        }
    } catch (IOException ex) {
        ex.printStackTrace();
    }

    button.setPreferredSize(new Dimension(width, height)); // Set preferred size for layout purposes
    button.setSize(width, height); // Set the button size directly
    return button;
}
```

```java
private void loadBackgroundImage() {
    try (InputStream imageStream = getClass().getResourceAsStream("/game/image/B.jpg")) {
        if (imageStream != null) {
            backgroundImage = ImageIO.read(imageStream);
        } else {
            System.err.println("Background image not found. Please check the path.");
        }
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}
```

Composition in the MainMenu class uses a combination of objects. It is the main component of UI makes it possible to design the appearance of menus and customize the behavior of buttons.

```java
public class MainMenu extends JPanel {

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);

        // Draw background image
        if (backgroundImage != null) {
            g.drawImage(backgroundImage, 0, 0, getWidth(), getHeight(), this);
        } else {
            g.setColor(Color.DARK_GRAY);
            g.fillRect(0, 0, getWidth(), getHeight());
        }

        // Center the buttons dynamically
        int panelWidth = getWidth();
        int panelHeight = getHeight();

        int startX = (panelWidth - buttonWidth) / 2;
        int startY = (panelHeight - buttonHeight) / 2 + 20; // Adjust to move slightly below center

        int exitX = startX;
        int exitY = startY + buttonHeight + 10; // Only a 5-pixel gap between buttons

        startButton.setBounds(startX, startY, buttonWidth, buttonHeight);
        exitButton.setBounds(exitX, exitY, buttonWidth, buttonHeight);
    }
}
```

Inheritance extends from Java Swing's JPanel, making it possible to use all of its functionality.

Polymorphism This method has been rewritten to draw a custom background instead of using JPanel's native drawing.

```java
// Create buttons with images
startButton = createImageButton("/game/image/START.png", startAction, buttonWidth, buttonHeight);
exitButton = createImageButton("/game/image/EXIT.png", exitAction, buttonWidth, buttonHeight);

add(titleLabel);
add(startButton);
add(exitButton);
```

Event Handling

ActionListener which is received as a parameter in the constructor and used to define the behavior of the startButton and exitButton buttons.

Chapter 3

Summary

Problems encountered during development

1 Destroyter Rock has some parts where it moves in place.

2 Destroyter Rock through the edge.

Unique program highlights

1 Items have been added, namely Bom and Blood.

2 Total points are accumulated and counting new scores when passing the checkpoint

Suggestions for teachers who want to explain

-