Space Invaders in Java

Submitted By:
Name: Kshitij Chandrakar
SAP: 500124827
Batch: 5
Course: B.Tech CSE (Spec. Data Science)
Semester: 4

Introduction and Problem Statement

This project is a simple implementation of the classic Space Invaders game, but in Java using Swing. It includes features such as player movement, shooting, alien movement, and a game over screen.

Github Link:

https://github.com/KshitijChandrakar/OOPS_Final_Project/

Some features

- Player movement (left and right)
- Player shooting
- Alien movement
- Alien shooting
- Collision detection
- High score display
- Game over screen

Requirements for playing

- Java Development Kit (JDK) 8 or higher
- Maven

Usage

- **Start the game**: Compile and Run the `Run` File
- **Move the player**: Use the left/right arrow keys or A and D keys to move the player.
- **Shoot**: Press the spacebar to shoot.
- **View high score**: Click the "High Score" button in the main menu.
- **View controls**: Click the "Controls" button in the main menu.
- **Quit the game**: Click the "Quit" button in the main menu or on the game over screen.

Project Structure

- **src/**
  - **score.txt**: All scores.
  - **Images/**: Contains all the image resources used in the game.
    - `alien.png`
    - `rocket.png`
    - `GameOver.png`
    - `ikona.png`
    - `logo.png`
  - **Font/**: Contains the custom font used in the Main Menu.
    - `font.ttf`
  - **Audio/**: Contains the custom sounds used in game.
  - `SPACE INVADERS SHOOT - Gaming Sound Effects HD FREE NO Copyright.wav` **UnitTest/**: Contains tests for some classes.
- **src/main/java/**
  - Game/**:Contains classes that makes the game .
    - **MainFrame.java**: The main frame that initializes the game (Main menu).
    - **MainMenuPanel.java**: The main menu panel with buttons to start the game, show the high score, show controls and quit the game.
    - **AlienShot.java**: Represents a shot fired by an alien.
    - **GameOver.java**: Frame displayed at the end of the game with options to return to the menu or quit.
    - **Sprite.java**: Base class for all moving objects in the game (aliens and player).
    - **Player.java**: Represents the player's character.
    - **Alien.java**: Represents an alien enemy.
    - **Shot.java**: Represents a shot fired by the player.
    - **GameOver.java** : Frame for game over.
  - **MainMenu/**:Contains menu buttons, panel, and frame. Here is Main class too.
    - **HighScoreButton.java**: Button to display the high score from a text file.
    - **PlayButton.java**: Button to start the game.
    - **OptionsButton.java**: Button to show the game controls.
    - **ExitButton.java**: Button to quit the game.
    - **MenuButton.java**: Base class for menu buttons.
  - **Handlers/**: Handlers for keyboard inputs, score writing and sound playing.
    - **KeyHandler.java**: Handles keyboard input.
    - **SoundManagers.java**: Handles sounds.
    - **ScoreManager.java**: Writes scores to score.txt.
  - **vendor/**:Contains not my code.
    - **BasicBlocks.java**:Represents houses on game screen(GamePanel).

Description of Game Classes

Sprite Class

The `Sprite` class is the base class for all moving objects in the game. It contains common properties and methods.

Properties: - `x`, `y`: Position of the sprite. - `width`, `height`: Dimensions of the sprite. - `image`: Image representing the sprite. - `destroyed`: Boolean indicating if the sprite is destroyed.

Methods: - `draw(Graphics2D g)`: Draws the sprite on the screen. - `getBounds()`: Returns a rectangle bounding the sprite. - `checkCollision(int shotX, int shotY, int tileSize)`: Checks if the sprite collides with a shot.

Player Class

The `Player` class represents the player's character. It extends the `Sprite` class and adds specific properties and methods.

Properties: - `lives`: Number of lives the player has. - `playerSpeed`: Speed of the player's movement.

Methods: - `moveLeft(int speed)`: Moves the player to the left. - `moveRight(int speed, int maxWidth, int tileSize)`: Moves the player to the right. - `playerMoving(KeyHandler keyHandler, int panelWidth, int tileSize, int speed)`: Moves the player based on keyboard input.

Alien Class

The `Alien` class represents an alien enemy. It extends the `Sprite` class and adds specific properties and methods.

Properties: - `speed`: Speed of the alien's movement.

Methods: - `move(CopyOnWriteArrayList<Alien> aliens, int panelWidth)`: Moves all aliens and checks for direction change. - `update(Shot shot, int tileSize)`: Updates the alien's state based on collisions with shots. - `shoot()`: Creates and returns a new `AlienShot`. - `checkCollision(int shotX, int shotY, int tileSize)`: Checks for collisions with player shots.

Shot Class

The `Shot` class represents a shot fired by the player.

Properties: - `x`, `y`: Position of the shot. - `isShooting`: Boolean indicating if the shot is currently active.

Methods: - `draw(Graphics2D g, int width, int height)`: Draws the shot on the screen. - `shooting(KeyHandler keyHandler, Player player, int shotSpeed)`: Updates the shot's position based on player input.

The `AlienShot` class represents a shot fired by an alien. It extends the `Shot` class and adds specific methods.

Methods: - `move(int shotSpeed)`: Moves the shot down the screen.

GamePanel Class

Properties

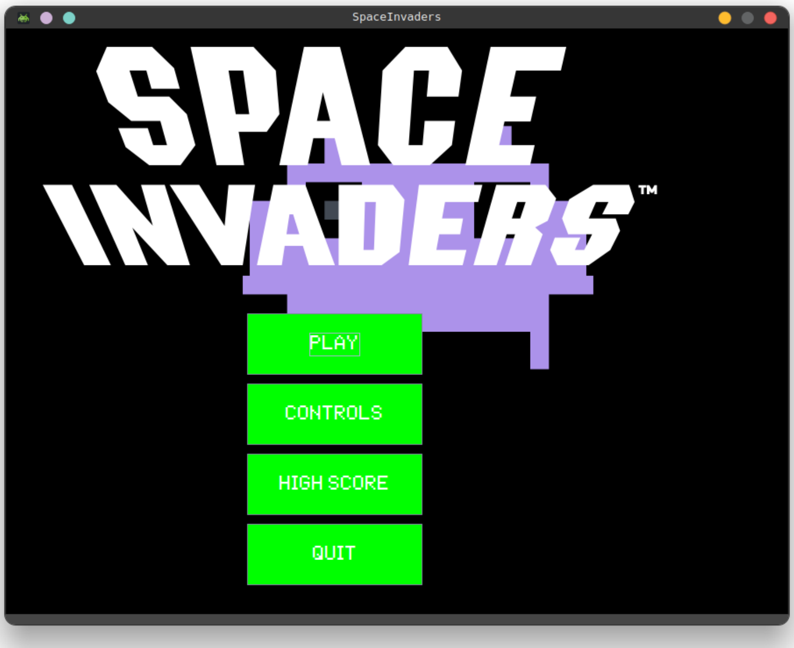
- **originalTileSize**: The original size of a tile (16 pixels).
- **scale**: The scale factor for resizing the tiles (3).
- **tileSize**: The scaled size of a tile.
- **maxScreenCol**: The maximum number of columns on the screen (16).
- **maxScreenRow**: The maximum number of rows on the screen (12).
- **width**: The width of the game screen.
- **height**: The height of the game screen.
- **FPS**: Frames per second (60).
- **gameThread**: The main game thread.
- **keyHandler**: Handles keyboard inputs.
- **soundManager**: Manages game sounds.
- **isGameOver**: Indicates if the game is over.
- **shotSpeed**: The speed of the player's shot.
- **playerSpeed**: The speed of the player.
- **alienSpeed**: The speed of the aliens.
- **player**: The player's character.
- **shot**: The player's shot.
- **numberOfDestroyedAliens**: The number of destroyed aliens.
- **bb**: The basic blocks in the game.
- **aliens**: A list of aliens.
- **alienShots**: A list of shots fired by aliens.
- **playerX**: The initial X position of the player.
- **playerY**: The initial Y position of the player.
- **alienShotTimer**: Timer for alien shots.
- **score**: The player's score.
- **gameFrame**: The game frame.
- **scoreManager**: Manages the score system and stores high scores.

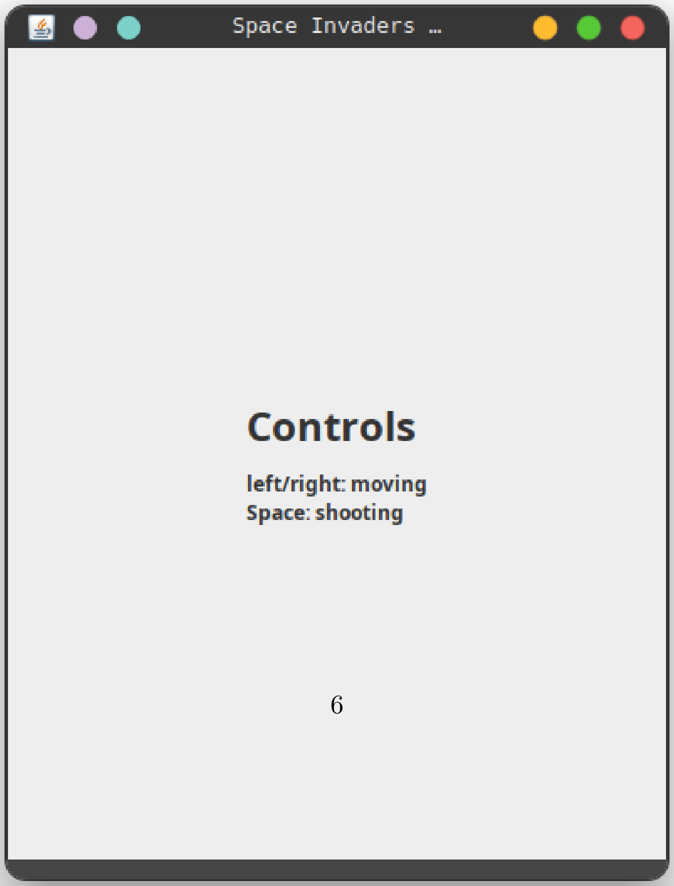
Methods

- **GamePanel()**: Constructor that initializes the game panel.
- **startGameThread()**: Starts the main game thread.
- **gameOver()**: Handles the game over state.
- **run()**: Main game loop.
- **update()**: Updates the game state.
- **checkBlockCollision(Shot shot)**: Checks collision between a shot and blocks.

- **checkBlockCollision(AlienShot alienShot)**: Checks collision between an alien shot and blocks.
- **createEnemies()**: Creates the initial set of aliens.
- **alienShoot()**: Handles alien shooting.
- **checkCollision(AlienShot alienShot, Player player)**: Checks collision between an alien shot and the player.
- **paintComponent(Graphics graphics)**: Renders the game components.

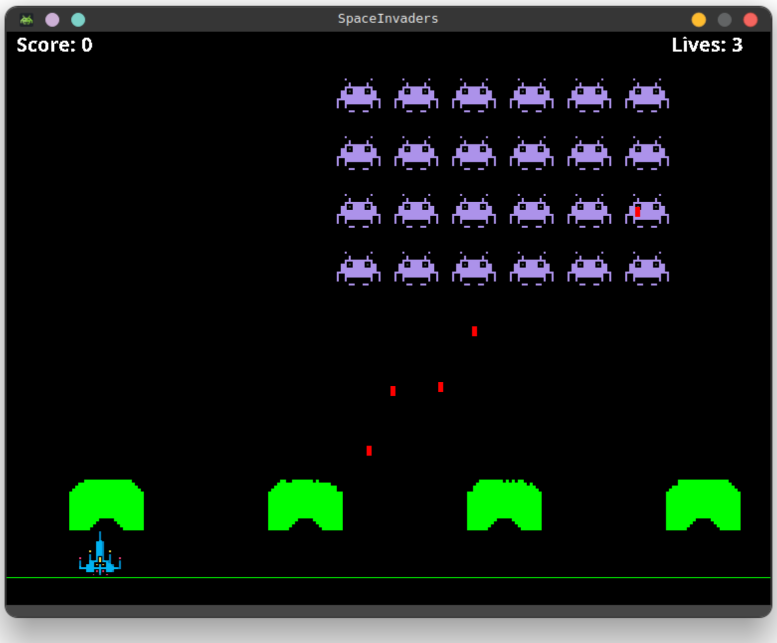
Screenshots

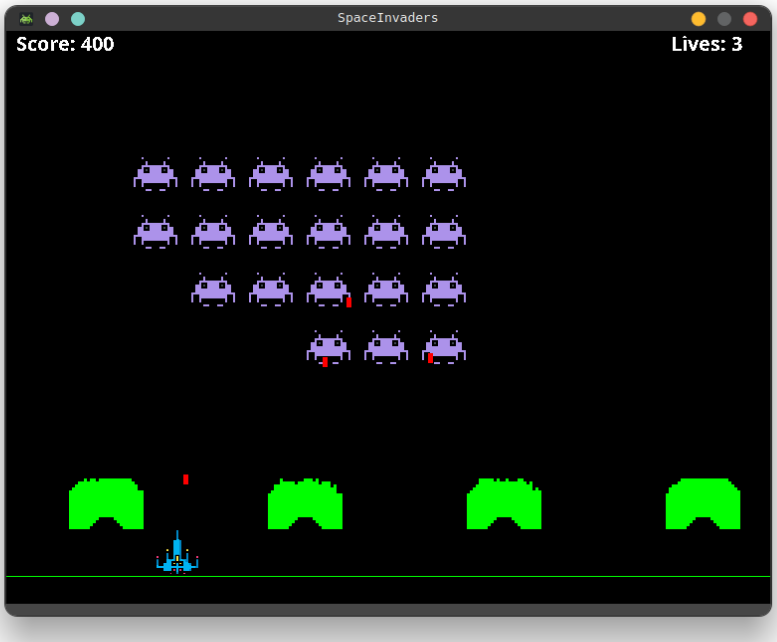


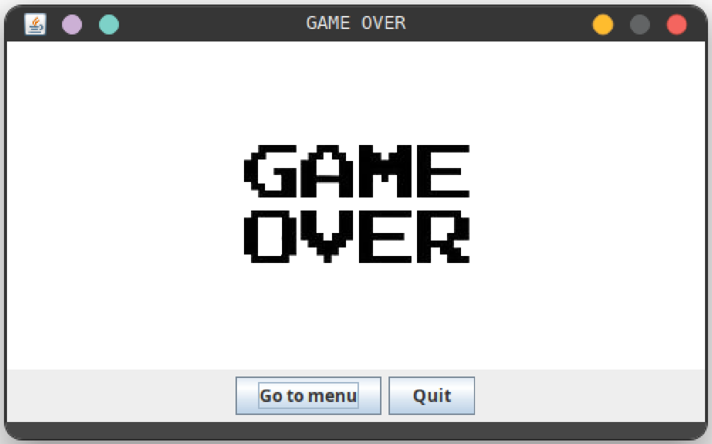


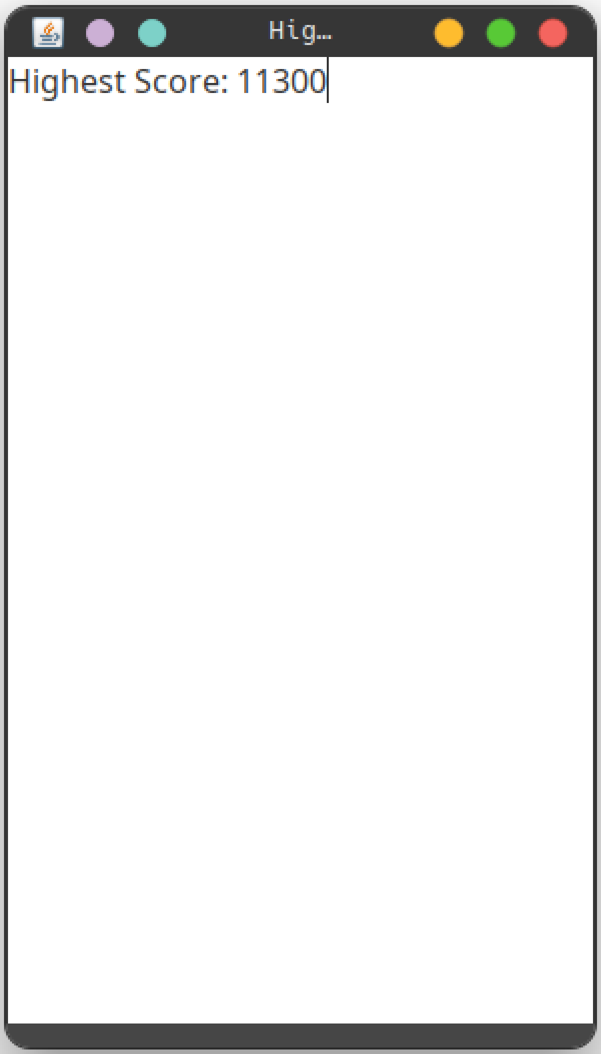
Controls

left/right: moving
Space: shooting


SpaceInvaders
Score: 0
Lives: 3


SpaceInvaders
Score: 400
Lives: 3


GAME OVER
GAME OVER
Go to menu
Quit



Highest Score: 11300

Code

/src/main/java ** ./Game/Alien.java **

```
package Game;

import javax.swing.*;
import java.awt.*;
import java.util.concurrent.CopyOnWriteArrayList;

public class Alien extends Sprite {
    static int speed = 2;

    public Alien(int x, int y, int width, int height) {
        super(x, y, width, height, "src/Images/alien.png");
    }

    public static void move(CopyOnWriteArrayList<Alien> aliens, int panelWidth) {
        boolean reverseDirection = false;

        for (Alien alien : aliens) {
            alien.x += speed;
            if (alien.x <= 0 || alien.x >= panelWidth - alien.width) {
                reverseDirection = true;
            }
        }

        if (reverseDirection) {
            speed *= -1;
            for (Alien alien : aliens) {
                alien.y += 20;
            }
        }
    }

    public void update(Shot shot, int tileSize) {
        if (shot.isShooting && checkCollision(shot.getX(), shot.getY(), tileSize)) {
            destroyed = true;
        }
    }

    public AlienShot shoot() {
        return new AlienShot(x + width / 2, y + height);
    }
}
```

** ./Game/AlienShot.java **

```
package Game;

import javax.swing.*;
```

```java
import java.awt.*;

public class AlienShot {
    int x;
    public int y;
    public int speed = 5;
    boolean isShooting = true;

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

    public void move() {
        y += speed;

    }

    public void draw(Graphics2D g) {
        g.setColor(Color.RED);
        g.fillRect(x, y, 5, 10);  // Example size of the alien shot
    }

    public Rectangle getBounds() {
        return new Rectangle(x, y, 5, 10);  // Example size of the alien shot
    }
}
```

** ./Game/GameFrame.java **

```java
package Game;

import javax.swing.*;

public class GameFrame extends JFrame {
    private GamePanel gamePanel;
    ImageIcon imageIcon=new ImageIcon("src/Images/ikona.png");
    public GameFrame(){
        gamePanel=new GamePanel();
        this.setTitle("SpaceInvaders");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setResizable(false);
        this.add(gamePanel);
        this.setIconImage(imageIcon.getImage());
        this.pack();
        this.setLocationRelativeTo(null);
        this.setVisible(true);
        gamePanel.startGameThread();
        //if(gamePanel.isGameOver()){
          //  this.dispose();
       // }

    }

}
```

** ./Game/GameOver.java **

```
package Game;

import MainMenu.MainFrame;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class GameOver extends JFrame {
    ImageIcon imageIcon = new ImageIcon("src/Images/GameOver.png");

    public GameOver() {
        setTitle("GAME OVER");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(500, 300);
        setResizable(false);
        setLocationRelativeTo(null);
        getContentPane().setBackground(Color.WHITE);

        // Vytvoření JLabel pro obrázek Game Over
        JLabel imageLabel = new JLabel(imageIcon);
        imageLabel.setHorizontalAlignment(JLabel.CENTER);
        imageLabel.setVerticalAlignment(JLabel.CENTER);
        add(imageLabel, BorderLayout.CENTER);

        // Vytvoření panelu pro tlačítka
        JPanel buttonPanel = new JPanel();
        buttonPanel.setLayout(new FlowLayout(FlowLayout.CENTER));

        // Tlačítko "Go to menu"
        JButton playAgainButton = new JButton("Go to menu");
        playAgainButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                // Zavřít okno Game Over
                dispose();
                // Spustit hru znovu
                MainFrame m = new MainFrame();
            }
        });
        buttonPanel.add(playAgainButton);

        // Tlačítko "Quit"
        JButton quitButton = new JButton("Quit");
        quitButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                // Ukončit aplikaci
                System.exit(0);
            }
        });
        buttonPanel.add(quitButton);

        // Přidat panel s tlačítky do okna
        add(buttonPanel, BorderLayout.SOUTH);

        setVisible(true);
    }


}
```

** ./Game/GamePanel.java **

```
package Game;

import Handlers.KeyHandler;
import Handlers.ScoreManager;
import Handlers.SoundManager;
import vendor.BasicBlocks;

import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Random;

import java.util.concurrent.CopyOnWriteArrayList;

public class GamePanel extends JPanel implements Runnable {

    final int originalTileSize = 16;
```

```java
final int scale = 3;
final int tileSize = originalTileSize * scale;
final int maxScreenCol = 16;
final int maxScreenRow = 12;
final int width = tileSize * maxScreenCol;
final int height = tileSize * maxScreenRow;
int FPS = 60;

Thread gameThread;
KeyHandler keyHandler = new KeyHandler();
SoundManager soundManager = new SoundManager();
private boolean playerAlive;
private boolean GameOver;

int shotSpeed = 10;
int playerSpeed = 3;
int alienSpeed = 2;
private Player player;
private Shot shot;
int numberOfDestroyedAliens = 0;

private BasicBlocks bb;
private CopyOnWriteArrayList<Alien> aliens;
private CopyOnWriteArrayList<AlienShot> alienShots;
private int playerX = 100;
private int playerY = 500;
private Timer alienShotTimer;
private int score = 0; // Score variable

private GameFrame gameFrame;
ScoreManager scoreManager;


public GamePanel() {

    this.setPreferredSize(new Dimension(width, height));
    this.setBackground(Color.black);
    this.setDoubleBuffered(true);
    this.addKeyListener(keyHandler);
    this.setFocusable(true);
    this.scoreManager = new ScoreManager("score.txt");

    aliens = new CopyOnWriteArrayList<>();
    shot = new Shot();
    createEnemies();
    bb = new BasicBlocks();
```

```java
        player = new Player(playerX, playerY, tileSize, tileSize);
        alienShots = new CopyOnWriteArrayList<>();

        // Set up a timer to shoot aliens every second
        alienShotTimer = new Timer(200, e -> alienShoot());
        alienShotTimer.start();



    }

    public void startGameThread() {
        gameThread = new Thread(this);
        gameThread.start();
    }

    public void gameOver() {
        scoreManager.writeScore(score);
        GameOver gameOver = new GameOver();
        gameThread.stop();
        alienShotTimer.stop();

    }


    @Override
    public void run() {
        double drawInterval = 1000000000 / FPS;
        double nextDrawTime = System.nanoTime() + drawInterval;
        while (gameThread != null) {
            update();
            repaint();
            try {
                double remainingTime = nextDrawTime - System.nanoTime();
                remainingTime = remainingTime / 1000000;
                if (remainingTime < 0) {
                    remainingTime = 0;
                }
                Thread.sleep((long) remainingTime);
                nextDrawTime += drawInterval;
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
```

```
public void update() {
    setPlayerAlive(player.isAlive());
    if (!player.isAlive()) {
        SwingUtilities.getWindowAncestor(this).dispose();
        gameOver();
        return;
    }

    player.playerMoving(keyHandler, width, tileSize, playerSpeed);
    shot.shooting(keyHandler, player, shotSpeed);

    if (!shot.isShooting) {
        soundManager.playShoot();
    }

    // Use CopyOnWriteArrayList to remove aliens while iterating
    for (Alien alien : aliens) {
        alien.update(shot, tileSize);
        if (alien.isDestroyed()) {
            aliens.remove(alien);
            shot.isShooting = false;
            numberOfDestroyedAliens++;
            score += 100; // Increment score by 100 for each destroyed alien
        }
        if (alien.getY() == player.getY()) {
            player.loseLife();
            if (!player.isAlive()) {
                SwingUtilities.getWindowAncestor(this).dispose();
                gameOver();
                return;
            }
        }

        // Check collision between aliens and blocks
        Iterator<Rectangle> blockIterator = bb.wall.iterator();
        while (blockIterator.hasNext()) {
            Rectangle block = blockIterator.next();
            if (alien.getBounds().intersects(block)) {
                blockIterator.remove();
            }
        }
    }

    if (numberOfDestroyedAliens == 24) {
        Alien.speed = Math.abs(Alien.speed) + 1;
        createEnemies();
        numberOfDestroyedAliens = 0;
    }

    Alien.move(aliens, width);

    for (AlienShot alienShot : alienShots) {
        alienShot.move();
        if (!alienShot.isShooting) {
            alienShots.remove(alienShot);
        } else if (checkCollision(alienShot, player)) {
            player.loseLife();
            if (!player.isAlive()) {
                SwingUtilities.getWindowAncestor(this).dispose();
                gameOver();
                return;
            }
            alienShots.remove(alienShot);
        } else if (checkBlockCollision(alienShot)) {
            alienShots.remove(alienShot);
        }
    }

    // Check collision with blocks for player shot
    if (shot.isShooting) {
        if (checkBlockCollision(shot)) {
            shot.isShooting = false;
        }
    }
}



private boolean checkBlockCollision(Shot shot) {
    Iterator<Rectangle> iterator = bb.wall.iterator();
    while (iterator.hasNext()) {
        Rectangle block = iterator.next();
        if (shot.getBounds(tileSize, tileSize).intersects(block)) {
            iterator.remove();

            return true;
        }
    }
    return false;
}
public void setPlayerAlive(boolean alive) {
    this.playerAlive = alive;
```

```
}

public boolean isPlayerAlive() {
    return playerAlive;
}

private boolean checkBlockCollision(AlienShot alienShot) {
    Iterator<Rectangle> iterator = bb.wall.iterator();
    while (iterator.hasNext()) {
        Rectangle block = iterator.next();
        if (alienShot.getBounds().intersects(block)) {
            iterator.remove();

            return true;
        }
    }
    return false;
}

public void createEnemies() {
    int startX = 0;
    int startY = 0;
    int gap = 10;
    int rows = 4;
    int cols = 6;

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            int x = startX + (tileSize + gap) * j;
            int y = startY + (tileSize + gap) * i;
            aliens.add(new Alien(x, y, tileSize, tileSize));
        }
    }
}

public void alienShoot() {
    if (!aliens.isEmpty()) {
        Random rand = new Random();
        Alien shootingAlien = aliens.get(rand.nextInt(aliens.size()));
        alienShots.add(shootingAlien.shoot());

    }
}

public boolean checkCollision(AlienShot alienShot, Player player) {
    return alienShot.getBounds().intersects(new Rectangle(player.getX(), player.getY(),
    }


    public void paintComponent(Graphics graphics) {
        super.paintComponent(graphics);
        Graphics2D g2d = (Graphics2D) graphics;
        player.draw(g2d);
        ArrayList<Alien> aliensCopy = new ArrayList<>(aliens);
        bb.draw(g2d);

        for (Alien alien : aliensCopy) {
            alien.draw(g2d);
        }

        if (shot.isShooting) {
            g2d.drawImage(new ImageIcon("src/Images/bullet.png").getImage(), shot.getX(), sh
        }

        for (AlienShot alienShot : alienShots) {
            alienShot.draw(g2d);
        }

        g2d.setColor(Color.GREEN);
        g2d.drawLine(0, player.getY() + tileSize, 9999, player.getY() + tileSize);

        // Draw the score
        g2d.setColor(Color.WHITE);
        g2d.setFont(new Font("Arial", Font.BOLD, 20));
        g2d.drawString("Score: " + score, 10, 20);

        // Draw the player's lives
        g2d.setColor(Color.WHITE);
        g2d.drawString("Lives: " + player.getLives(), width - 100, 20);

        g2d.dispose();
    }


    public boolean isGameOver() {
        return !playerAlive;
    }
}
```

** ./Game/Main.java **

```
package Game;
```

```java
import MainMenu.MainFrame;

public class Main {
    public static void main(String[] args) {
        System.out.println("Hello world!");
        MainFrame mainFrame=new MainFrame();
    }
}
```

** ./Game/Player.java **

```java
package Game;

import Handlers.KeyHandler;

public class Player extends Sprite {
    private int lives;

    public Player(int x, int y, int width, int height) {
        super(x, y, width, height, "src/Images/rocket.png");
        this.lives = 3;
    }

    public int getLives() {
        return lives;
    }

    public boolean isAlive() {
        return lives > 0;
    }

    public void loseLife() {
        if (lives > 0) {
            lives--;
        }
    }

    public void moveLeft(int speed) {
        x -= speed;
        if (x < 0) {
            x = 0;
        }
    }

    public void moveRight(int speed, int maxWidth, int tileSize) {
        x += speed;
        if (x <= 0) {
            x = 0;
        } else if (x >= maxWidth - tileSize) {
            x = maxWidth - tileSize;
        }
    }

    public void playerMoving(KeyHandler keyHandler, int panelWidth, int tileSize, int speed
        if (keyHandler.leftPressed) {
            moveLeft(speed);
        } else if (keyHandler.rightPressed) {
            moveRight(speed, panelWidth, tileSize);
        }
    }
}
```

** ./Game/Shot.java **

```
package Game;

import Handlers.KeyHandler;

import java.awt.*;

public class Shot {
    int x, y;
    boolean isShooting;

    public Shot() {
        isShooting = false;
    }

    public int getX() {
        return x;
    }

    public void setX(int x) {
        this.x = x;
    }

    public int getY() {
        return y;
    }

    public void setY(int y) {
        this.y = y;
    }
```

```java
    public Rectangle getBounds(int width, int height) {
        return new Rectangle(x, y, width, height);
    }

    public void draw(Graphics2D g, int width, int height) {
        g.setColor(Color.CYAN);
        g.fillRect(x, y, width, height);
    }

    public void shooting(KeyHandler keyHandler, Player player, int shotSpeed) {
        if (keyHandler.shotPressed) {
            if (!isShooting) {
                isShooting = true;
                this.x = player.getX();
                this.y = player.getY();
            }
        }
        if (isShooting) {
            this.y -= shotSpeed;
            if (this.y < 0) {
                isShooting = false;
            }
        }
    }
}
```

** ./Game/Sprite.java **

```java
package Game;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.concurrent.CopyOnWriteArrayList;
public class Sprite {
    protected int x, y, width, height;
    public Image image;
    protected boolean destroyed;

    public Sprite(int x, int y, int width, int height, String imagePath) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.image = new ImageIcon(imagePath).getImage();
        this.destroyed = false;
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    public int getWidth() {
        return width;
    }

    public int getHeight() {
        return height;
    }

    public boolean isDestroyed() {
        return destroyed;
    }

    public void draw(Graphics2D g) {
        if (!destroyed) {
            g.drawImage(image, x, y, width, height, null);
        }
    }

    public Rectangle getBounds() {
        return new Rectangle(x, y, width, height);
    }

    public void setDestroyed(boolean destroyed) {
        this.destroyed = destroyed;
    }

    public boolean checkCollision(int shotX, int shotY, int tileSize) {
        Rectangle spriteRect = new Rectangle(x, y, width, height);
        Rectangle shotRect = new Rectangle(shotX, shotY, tileSize, tileSize);
        return spriteRect.intersects(shotRect);
    }
}
```

** ./Handlers/KeyHandler.java **

```
package Handlers;
```

```java
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;

import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;

public class KeyHandler implements KeyListener {

    public boolean leftPressed, rightPressed, shotPressed, menuPressed;




    @Override
    public void keyTyped(KeyEvent e) {
    }

    @Override
    public void keyPressed(KeyEvent e) {
        int code = e.getKeyCode();
        if (code == KeyEvent.VK_A) {
            leftPressed = true;
        }
        if (code == KeyEvent.VK_D) {
            rightPressed = true;
        }
        if (code == KeyEvent.VK_LEFT) {
            leftPressed = true;
        }
        if (code == KeyEvent.VK_RIGHT) {
            rightPressed = true;
        }
        if (code == KeyEvent.VK_SPACE) {
            shotPressed = true;
        }
        if (code == KeyEvent.VK_ESCAPE) {
            menuPressed = true;

        }

    }

    @Override
    public void keyReleased(KeyEvent e) {
        int code = e.getKeyCode();
        if (code == KeyEvent.VK_A) {
            leftPressed = false;
        }
        if (code == KeyEvent.VK_D) {
            rightPressed = false;
        }
        if (code == KeyEvent.VK_LEFT) {
            leftPressed = false;
        }
        if (code == KeyEvent.VK_RIGHT) {
            rightPressed = false;
        }
        if (code == KeyEvent.VK_SPACE) {
            shotPressed = false;

        }
        if (code == KeyEvent.VK_ESCAPE) {
            menuPressed = false;
        }

    }
}
```

** ./Handlers/ScoreManager.java **

```
package Handlers;

import java.io.FileWriter;
import java.io.IOException;

public class ScoreManager {

    private String fileName;

    public ScoreManager(String fileName) {
        this.fileName = fileName;
    }

    public void writeScore(int score) {
        try (FileWriter writer = new FileWriter(fileName, true)) {
            writer.write(score + System.lineSeparator());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

** ./Handlers/SoundManager.java **

```
package Handlers;

import javax.sound.sampled.*;
import java.io.File;
import java.io.IOException;

import javax.sound.sampled.*;
import java.io.File;
import java.io.IOException;

public class SoundManager {
    private Clip shootClip;
    private Clip gameSoundClip;
    private Clip collisionClip;

    public SoundManager() {
        shootClip = loadSound("src/Audio/SPACE INVADERS SHOOT - Gaming Sound Effects HD FREE


    }

    private Clip loadSound(String filePath) {
        try {
            File file = new File(filePath);
            AudioInputStream audioInputStream = AudioSystem.getAudioInputStream(file);
            Clip clip = AudioSystem.getClip();
            clip.open(audioInputStream);
            return clip;
        } catch (UnsupportedAudioFileException | IOException | LineUnavailableException e) {
            e.printStackTrace();
            return null;
        }
    }

    public void playShoot() {
        playClip(shootClip);
    }






    private void playClip(Clip clip) {
        if (clip != null) {
            clip.stop();
            clip.setFramePosition(0);
            clip.start();
        }
    }
}
```

** ./MainMenu/ExitButton.java **

```
package MainMenu;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class ExitButton extends MenuButton {
    public ExitButton(String text, Font font,int number) {
        super(text, font,number);
    }

    @Override
    public void action(MainMenuPanel m) {
        this.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.exit(0);
            }
        });
    }
}
```

** ./MainMenu/HighScoreButton.java **

```
package MainMenu;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class HighScoreButton extends MenuButton {
    public HighScoreButton(String text, Font font, int number) {
        super(text, font, number);
        addActionListener();
    }
    private void addActionListener() {
        this.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                JFrame highScoreFrame = new JFrame("High Score");
                highScoreFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
                highScoreFrame.setSize(300, 200);
                highScoreFrame.setLocationRelativeTo(null);

                JTextArea highScoreTextArea = new JTextArea();
                highScoreTextArea.setEditable(false);
                highScoreTextArea.setFont(new Font("Arial", Font.PLAIN, 16));

                try (BufferedReader reader = new BufferedReader(new FileReader("score.txt"))
                    String line;
                    int highestScore = Integer.MIN_VALUE;
                    while ((line = reader.readLine()) != null) {
                        if (!line.trim().isEmpty()) {
                            int score = Integer.parseInt(line);
                            if (score > highestScore) {
                                highestScore = score;
                            }
                        }
                    }
                    highScoreTextArea.setText("Highest Score: " + (highestScore == Integer.M
                } catch (IOException ex) {
                    ex.printStackTrace();
                    highScoreTextArea.setText("Error loading high score.");
                }

                highScoreFrame.add(highScoreTextArea);
                highScoreFrame.setVisible(true);
            }
        });
    }

    @Override
    public void action(MainMenuPanel m) {

    }
}
```

** ./MainMenu/MainFrame.java **

```
package MainMenu;
```

```java
import javax.swing.*;

public class MainFrame extends JFrame {
    ImageIcon image = new ImageIcon("src/Images/ikona.png");
    MainMenuPanel mainMenuPanel = new MainMenuPanel();

    public MainFrame() {
        this.add(mainMenuPanel);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setResizable(false);
        this.setTitle("SpaceInvaders");
        this.setIconImage(image.getImage());
        this.setBounds(100, 100, 900, 700);
        this.setVisible(true);
        if(!mainMenuPanel.isVisible()){
            this.setVisible(false);
        }
    }
    public static void main(String[] args) {
        System.out.println("Hello world!");
        MainFrame mainFrame=new MainFrame();
    }
}
```

** ./MainMenu/MainMenuPanel.java **

```java
package MainMenu;

import javax.swing.*;
import java.awt.*;
import java.io.File;
import java.io.IOException;

public class MainMenuPanel extends JPanel implements Runnable {
    final int width = 750;
    final int height = 750;
    Thread thread;

    int FPS = 60;
    Font f;


    int alienX = 0;
    int alienY = 100;
    int alienSpeed = 5;
    boolean movingRight = true;
```

```
{
    try {
        f = Font.createFont(Font.TRUETYPE_FONT, new File("src/Font/font.ttf")).deriveFon
    } catch (FontFormatException | IOException e) {
        throw new RuntimeException(e);
    }
}

ImageIcon logoImage = new ImageIcon("src/Images/logo.png");
ImageIcon alienImage = new ImageIcon("src/Images/alien.png");
JLabel logoLabel = new JLabel(logoImage);
JLabel alienLabel = new JLabel(alienImage);
PlayButton playButton = new PlayButton("Play", f, 0);
OptionsButton optionsButton = new OptionsButton("Controls", f, 0);
HighScoreButton highScoreButton = new HighScoreButton("High Score", f, 180);
ExitButton exitButton = new ExitButton("Quit", f, 0);

public MainMenuPanel() {
    this.setPreferredSize(new Dimension(width, height));
    this.setBackground(Color.BLACK);
    this.setDoubleBuffered(true);
    setLayout(null);


    // Setting location and size for the logo label
    logoLabel.setBounds((width - logoImage.getIconWidth()) , 20, logoImage.getIconWidth(
    add(logoLabel);



    // Setting fixed size for buttons
    Dimension buttonSize = new Dimension(200, 70);
    playButton.setPreferredSize(buttonSize);
    optionsButton.setPreferredSize(buttonSize);
    highScoreButton.setPreferredSize(buttonSize);
    exitButton.setPreferredSize(buttonSize);

    // Adding buttons with absolute positioning
    playButton.setBounds(width / 2 - 100, height / 2 - 50, 200, 70);
    optionsButton.setBounds(width / 2 - 100, height / 2 + 30, 200, 70);
    highScoreButton.setBounds(width / 2 - 100, height / 2 + 110, 200, 70);
    exitButton.setBounds(width / 2 - 100, height / 2 + 190, 200, 70);

    add(playButton);
    add(optionsButton);
    add(highScoreButton);
```

```
        add(exitButton);
        // Setting location and size for the alien label
        alienLabel.setBounds(alienX, alienY, 400, 300);
        add(alienLabel);

        // Setting up button actions
        playButton.action(this);
        optionsButton.action(this);
        highScoreButton.action(this);
        exitButton.action(this);


        // Start the animation thread
        thread = new Thread(this);
        thread.start();
    }

    @Override
    public void run() {
        double drawInterval = 1000000000 / FPS;
        double nextDrawTime = System.nanoTime() + drawInterval;
        while (thread != null) {
            update();
            repaint();
            try {
                double remainingTime = nextDrawTime - System.nanoTime();
                remainingTime = remainingTime / 1000000;
                if (remainingTime < 0) {
                    remainingTime = 0;
                }
                Thread.sleep((long) remainingTime);
                nextDrawTime += drawInterval;
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    public void update() {
        alienLabel.setLocation(alienX, alienY);
        if (movingRight) {
            alienX += alienSpeed;
            if (alienX + alienLabel.getWidth() >= width) {
                movingRight = false;
            }
        } else {
            alienX -= alienSpeed;
            if (alienX <= 0) {
                movingRight = true;
            }
        }

    }

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

    }

}
```

** ./MainMenu/MenuButton.java **

```
package MainMenu;

import MainMenu.MainMenuPanel;

import javax.swing.*;
import java.awt.*;
import java.io.IOException;

public abstract class MenuButton extends JButton {
    final int width=200;
    final int height=100;

    public MenuButton(String text,Font font,int number){
        this.setBounds(250,300+number,width,height);
        this.setText(text);
        this.setBackground(Color.green);
        this.setForeground(Color.WHITE);
        this.setFont(font);
    }

    public abstract void action(MainMenuPanel m) throws IOException, FontFormatException;

}
```

** ./MainMenu/MenuLabel.java **

```
package MainMenu;

import javax.swing.*;
```

```java
public class MenuLabel extends JLabel {
    public MenuLabel(ImageIcon icon){
        this.setVerticalAlignment(JLabel.TOP);
        this.setHorizontalAlignment(JLabel.CENTER);
        this.setIcon(icon);
    }
}
```

** ./MainMenu/OptionsButton.java **

```java
package MainMenu;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class OptionsButton extends MenuButton {
    public OptionsButton(String text, Font font,int number) {
        super(text, font,number);

    }

    @Override
    public void action(MainMenuPanel m) {
        this.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {//creating window for controls
                JFrame frame = new JFrame("Space Invaders - Controls");
                JLabel label = new JLabel("<html><h1>Controls</h1>"
                        + "<p>left/right: moving</p>"
                        + "<p>Space: shooting</p></html>");
                label.setHorizontalAlignment(CENTER);

                frame.getContentPane().add(label);

                frame.setSize(400, 200);
                frame.setLocationRelativeTo(null);
                frame.setVisible(true);
            }
        });
    }
}
```

** ./MainMenu/PlayButton.java **

```java
package MainMenu;

import Game.GameFrame;
import Game.GamePanel;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class PlayButton extends MenuButton {
    public PlayButton(String text, Font font, int number) {
        super(text, font, number);
    }

    @Override
    public void action(MainMenuPanel m) {
        this.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                GamePanel panel = new GamePanel();

                SwingUtilities.getWindowAncestor(m).dispose();

                GameFrame gameFrame= new GameFrame();
            }
        });
    }
}
```

** ./vendor/BasicBlocks.java **

```
package vendor;

import java.awt.*;
import java.util.ArrayList;

import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.Rectangle;
import java.util.ArrayList;

public class BasicBlocks {

    public ArrayList<Rectangle> wall = new ArrayList<Rectangle>();

    public BasicBlocks(){
        basicBlocks(75, 450);
        basicBlocks(275, 450);
        basicBlocks(475, 450);
        basicBlocks(675, 450);
    }

    public void draw(Graphics2D g){
        g.setColor(Color.GREEN);
        for(int i = 0; i < wall.size(); i++){
            g.fill(wall.get(i));
        }
    }

    public void basicBlocks(int xPos, int yPos){
        int wallWidth = 3;
        int x = 0;
        int y = 0;

        for(int i = 0; i < 13; i++){
            if((14 + (i * 2) + wallWidth < 22 + wallWidth)){
                row(14 + (i * 2) + wallWidth, xPos - (i * 3), yPos + (i * 3));
                x = (i * 3) + 3;
            }else{
                row(22 + wallWidth, xPos - x, yPos + (i * 3));
                y = (i * 3);
            }
        }

        //Left side.
        for(int i = 0; i < 5; i++){
            row(8 + wallWidth - i, xPos - x, (yPos + y) + (i * 3));
        }

        //Right side.
        for(int i = 0; i < 5; i++){
            row(8 + wallWidth - i, (xPos - x + (14 * 3)) + (i * 3), (yPos + y) + (i * 3));
        }
    }

    public void row(int rows, int xPos, int yPos){
        for(int i = 0; i < rows; i++){
            Rectangle brick = new Rectangle(xPos + (i * 3), yPos, 3, 3);
            wall.add(brick);
        }
    }

    public void reset(){
        wall.clear();
```

```
        basicBlocks(75, 450);
        basicBlocks(275, 450);
        basicBlocks(475, 450);
        basicBlocks(675, 450);
    }
}
```