import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

import java.io.\*;

import javax.sound.sampled.AudioInputStream;

import javax.sound.sampled.AudioSystem;

import javax.sound.sampled.Clip;

import javax.sound.sampled.LineUnavailableException;

import javax.sound.sampled.UnsupportedAudioFileException;

import java.io.IOException;

class SpaceInvadersGame extends JPanel {

private Image backgroundImage;

private Image playerRocketImage;

private List<Enemy> enemies;

private List<Bullet> bullets;

private List<EnemyBullet> enemyBullets;

private List<BigEnemy> bigEnemies;

private int score = 0;

private int level = 1;

private int lives = 3;

private boolean gameOver = false;

private boolean gameStarted = false;

private boolean playerRocketBlink = false;

private int enemyBulletSpeed = 2;

private Clip shootingSound;

private Clip enemyExplosionSound;

private Clip enteringSound;

private Clip shipLoseLifeSound;

private Clip gameOverSound;

public SpaceInvadersGame() {

backgroundImage = new ImageIcon("backgroundSkin.jpg").getImage();

playerRocketImage = new ImageIcon("shipSkin.gif").getImage();

enemies = new ArrayList<>();

bullets = new ArrayList<>();

enemyBullets = new ArrayList<>();

bigEnemies = new ArrayList<>();

for (int i = 0; i < 5; i++) {

int x = (int) (Math.random() \* 500);

int y = (int) (Math.random() \* 300);

enemies.add(new Enemy(x, y, true));

}

for (int i = 0; i < 2; i++) {

int x = (int) (Math.random() \* 500);

int y = (int) (Math.random() \* 300);

bigEnemies.add(new BigEnemy(x, y, true));

}

loadSoundEffects();

Timer timer = new Timer(16, new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

if (gameStarted && !gameOver) {

moveEnemiesHorizontally();

for (Enemy enemy : enemies) {

enemy.move();

if (enemy.canShoot()) {

enemyBullets.add(new EnemyBullet(enemy.getX() + enemy.getWidth() / 2, enemy.getY() + enemy.getHeight()));

enemy.setCanShoot(false);

}

}

for (BigEnemy bigEnemy : bigEnemies) {

bigEnemy.move();

if (bigEnemy.canShoot()) {

enemyBullets.add(new EnemyBullet(bigEnemy.getX() + bigEnemy.getWidth() / 2, bigEnemy.getY() + bigEnemy.getHeight()));

bigEnemy.setCanShoot(false);

}

}

moveBullets();

moveEnemyBullets();

checkCollisions();

repaint();

}

}

});

timer.start();

addKeyListener(new KeyAdapter() {

@Override

public void keyPressed(KeyEvent e) {

int keyCode = e.getKeyCode();

if (!gameStarted) {

startGame();

} else if (!gameOver) {

if (keyCode == KeyEvent.VK\_LEFT) {

Player.x -= 5;

} else if (keyCode == KeyEvent.VK\_RIGHT) {

Player.x += 5;

} else if (keyCode == KeyEvent.VK\_SPACE) {

bullets.add(new Bullet(Player.x + 20, Player.y));

playShootingSound();

}

}

}

});

setFocusable(true);

requestFocusInWindow();

}

private void loadSoundEffects() {

try {

AudioInputStream shootingStream = AudioSystem.getAudioInputStream(new File("bulletSound.wav"));

shootingSound = AudioSystem.getClip();

shootingSound.open(shootingStream);

AudioInputStream explosionStream = AudioSystem.getAudioInputStream(new File("damageSound.wav"));

enemyExplosionSound = AudioSystem.getClip();

enemyExplosionSound.open(explosionStream);

AudioInputStream enemyshootingStream = AudioSystem.getAudioInputStream(new File("alienBeam.wav"));

enteringSound = AudioSystem.getClip();

enteringSound.open(enemyshootingStream);

AudioInputStream shipLoseLifeStream = AudioSystem.getAudioInputStream(new File("hitmarkerSound.wav"));

shipLoseLifeSound = AudioSystem.getClip();

shipLoseLifeSound.open(shipLoseLifeStream);

AudioInputStream gameOverStream = AudioSystem.getAudioInputStream(new File("deathSound.wav"));

gameOverSound = AudioSystem.getClip();

gameOverSound.open(gameOverStream);

} catch (UnsupportedAudioFileException | LineUnavailableException | IOException e) {

e.printStackTrace();

}

}

private void playShootingSound() {

if (shootingSound.isRunning()) {

shootingSound.stop();

}

shootingSound.setFramePosition(0);

shootingSound.start();

}

private void playEnemyExplosionSound() {

if (enemyExplosionSound.isRunning()) {

enemyExplosionSound.stop();

}

enemyExplosionSound.setFramePosition(0);

enemyExplosionSound.start();

}

private void playEnteringSound() {

if (enteringSound.isRunning()) {

enteringSound.stop();

}

enteringSound.setFramePosition(0);

enteringSound.start();

}

private void playShipLoseLifeSound() {

if (shipLoseLifeSound.isRunning()) {

shipLoseLifeSound.stop();

}

shipLoseLifeSound.setFramePosition(0);

shipLoseLifeSound.start();

}

private void playGameOverSound() {

if (gameOverSound.isRunning()) {

gameOverSound.stop();

}

gameOverSound.setFramePosition(0);

gameOverSound.start();

}

public void startGame() {

gameStarted = true;

gameOver = false;

score = 0;

level = 1;

lives = 3;

enemies.clear();

bullets.clear();

enemyBullets.clear();

bigEnemies.clear();

for (int i = 0; i < 5; i++) {

int x = (int) (Math.random() \* 500);

int y = (int) (Math.random() \* 300);

enemies.add(new Enemy(x, y, true));

}

for (int i = 0; i < 2; i++) {

int x = (int) (Math.random() \* 500);

int y = (int) (Math.random() \* 300);

bigEnemies.add(new BigEnemy(x, y, true));

}

requestFocusInWindow();

playEnteringSound();

}

private void moveEnemiesHorizontally() {

for (Enemy enemy : enemies) {

int newX = enemy.getX() + enemy.getDirection();

if (newX < 0 || newX + enemy.getWidth() > getWidth()) {

enemy.reverseDirection();

}

enemy.setX(newX);

}

for (BigEnemy bigEnemy : bigEnemies) {

int newX = bigEnemy.getX() + bigEnemy.getDirection();

if (newX < 0 || newX + bigEnemy.getWidth() > getWidth()) {

bigEnemy.reverseDirection();

}

bigEnemy.setX(newX);

}

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

drawBackground(g);

drawPlayerRocket(g);

drawEnemies(g);

drawBigEnemies(g);

drawBullets(g);

drawEnemyBullets(g);

drawStats(g);

if (gameOver) {

g.setColor(Color.RED);

g.setFont(new Font("Arial", Font.BOLD, 40));

g.drawString("Game Over", 350, 300);

}

}

private void drawBackground(Graphics g) {

g.drawImage(backgroundImage, 0, 0, getWidth(), getHeight(), this);

}

private void drawPlayerRocket(Graphics g) {

if (!playerRocketBlink) {

g.drawImage(playerRocketImage, Player.x, Player.y, this);

}

}

private void drawEnemies(Graphics g) {

for (Enemy enemy : enemies) {

g.drawImage(enemy.getImage(), enemy.getX(), enemy.getY(), this);

}

}

private void drawBigEnemies(Graphics g) {

for (BigEnemy bigEnemy : bigEnemies) {

g.drawImage(bigEnemy.getImage(), bigEnemy.getX(), bigEnemy.getY(), this);

}

}

private void drawBullets(Graphics g) {

for (Bullet bullet : bullets) {

if (bullet.isActive()) {

g.setColor(Color.YELLOW);

g.fillRect((int) bullet.getX(), (int) bullet.getY(), 2, 10);

}

}

}

private void drawEnemyBullets(Graphics g) {

for (EnemyBullet bullet : enemyBullets) {

if (bullet.isActive()) {

g.setColor(Color.RED);

g.fillRect((int) bullet.getX(), (int) bullet.getY(), 2, 10);

}

}

}

private void drawStats(Graphics g) {

g.setColor(Color.WHITE);

g.setFont(new Font("Arial", Font.PLAIN, 20));

g.drawString("Score: " + score, 20, 30);

g.drawString("Level: " + level, 20, 60);

g.drawString("Lives: " + lives, 20, 90);

}

private void moveBullets() {

for (Bullet bullet : bullets) {

bullet.move();

}

bullets.removeIf(bullet -> !bullet.isActive());

}

private void moveEnemyBullets() {

for (EnemyBullet bullet : enemyBullets) {

bullet.move();

}

enemyBullets.removeIf(bullet -> !bullet.isActive());

}

private void checkCollisions() {

Iterator<Enemy> enemyIterator = enemies.iterator();

while (enemyIterator.hasNext()) {

Enemy enemy = enemyIterator.next();

int enemyX = enemy.getX();

int enemyY = enemy.getY();

int enemyWidth = enemy.getWidth();

int enemyHeight = enemy.getHeight();

Iterator<Bullet> bulletIterator = bullets.iterator();

while (bulletIterator.hasNext()) {

Bullet bullet = bulletIterator.next();

double bulletX = bullet.getX();

double bulletY = bullet.getY();

if (bulletX >= enemyX && bulletX <= enemyX + enemyWidth &&

bulletY >= enemyY && bulletY <= enemyY + enemyHeight) {

bulletIterator.remove();

enemyIterator.remove();

score += 10;

playEnemyExplosionSound();

if (enemies.isEmpty() && bigEnemies.isEmpty()) {

level++;

spawnNewEnemies();

spawnNewBigEnemies();

}

}

}

}

Iterator<BigEnemy> bigEnemyIterator = bigEnemies.iterator();

while (bigEnemyIterator.hasNext()) {

BigEnemy bigEnemy = bigEnemyIterator.next();

int bigEnemyX = bigEnemy.getX();

int bigEnemyY = bigEnemy.getY();

int bigEnemyWidth = bigEnemy.getWidth();

int bigEnemyHeight = bigEnemy.getHeight();

Iterator<Bullet> bulletIterator = bullets.iterator();

while (bulletIterator.hasNext()) {

Bullet bullet = bulletIterator.next();

double bulletX = bullet.getX();

double bulletY = bullet.getY();

if (bulletX >= bigEnemyX && bulletX <= bigEnemyX + bigEnemyWidth &&

bulletY >= bigEnemyY && bulletY <= bigEnemyY + bigEnemyHeight) {

bulletIterator.remove();

bigEnemyIterator.remove();

score += 20;

playEnemyExplosionSound();

if (enemies.isEmpty() && bigEnemies.isEmpty()) {

level++;

spawnNewEnemies();

spawnNewBigEnemies();

}

}

}

}

for (EnemyBullet bullet : enemyBullets) {

double bulletX = bullet.getX();

double bulletY = bullet.getY();

int playerX = Player.x;

int playerY = Player.y;

int playerWidth = playerRocketImage.getWidth(this);

int playerHeight = playerRocketImage.getHeight(this);

if (bulletX >= playerX && bulletX <= playerX + playerWidth &&

bulletY >= playerY && bulletY <= playerY + playerHeight) {

bullet.setActive(false);

lives--;

playShipLoseLifeSound();

if (lives <= 0) {

gameOver = true;

playGameOverSound();

} else {

blinkPlayerRocket();

Player.x = 500;

}

}

}

for (Enemy enemy : enemies) {

int enemyX = enemy.getX();

int enemyY = enemy.getY();

int enemyWidth = enemy.getWidth();

int enemyHeight = enemy.getHeight();

int playerX = Player.x;

int playerY = Player.y;

int playerWidth = playerRocketImage.getWidth(this);

int playerHeight = playerRocketImage.getHeight(this);

if (playerX + playerWidth >= enemyX && playerX <= enemyX + enemyWidth &&

playerY + playerHeight >= enemyY && playerY <= enemyY + enemyHeight) {

lives--;

if (lives <= 0) {

gameOver = true;

playGameOverSound();

} else {

blinkPlayerRocket();

Player.x = 500;

}

}

}

}

private void spawnNewEnemies() {

enemies.clear();

for (int i = 0; i < 5 + level; i++) {

int x = (int) (Math.random() \* 500);

int y = (int) (Math.random() \* 300);

enemies.add(new Enemy(x, y, true));

}

}

private void spawnNewBigEnemies() {

bigEnemies.clear();

for (int i = 0; i < 2; i++) {

int x = (int) (Math.random() \* 500);

int y = (int) (Math.random() \* 300);

bigEnemies.add(new BigEnemy(x, y, true));

}

}

private void blinkPlayerRocket() {

Timer timer = new Timer(200, new ActionListener() {

int blinkCount = 0;

@Override

public void actionPerformed(ActionEvent e) {

playerRocketBlink = !playerRocketBlink;

repaint();

blinkCount++;

if (blinkCount >= 6) {

((Timer) e.getSource()).stop();

playerRocketBlink = false;

}

}

});

timer.setRepeats(true);

timer.start();

}

public static void main(String[] args) {

SwingUtilities.invokeLater(new Runnable() {

@Override

public void run() {

JFrame frame = new JFrame("Space Invaders");

SpaceInvadersGame game = new SpaceInvadersGame();

frame.add(game);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setPreferredSize(new Dimension(800, 600));

frame.pack();

frame.setVisible(true);

frame.setResizable(false);

JButton startButton = new JButton("Start Game");

startButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

game.startGame();

}

});

frame.add(startButton, BorderLayout.SOUTH);

}

});

}

}

class Player {

static int x = 500;

static int y = 450;

}

class Bullet {

private double x;

private double y;

private boolean active = true;

public Bullet(double x, double y) {

this.x = x;

this.y = y;

}

public void move() {

y -= 5;

if (y < 0) {

active = false;

}

}

public double getX() {

return x;

}

public double getY() {

return y;

}

public boolean isActive() {

return active;

}

public void setActive(boolean active) {

this.active = active;

}

}

class Enemy {

private int x;

private int y;

private int direction; // Direction of horizontal movement (left or right)

private boolean canShoot = true;

private final Image image;

public Enemy(int x, int y, boolean initialDirection) {

this.x = x;

this.y = y;

this.direction = initialDirection ? 1 : -1;

image = new ImageIcon("alien1Skin.gif").getImage();

}

public void move() {

y += 1;

if (y > 600) {

y = 0;

x = (int) (Math.random() \* 500);

}

}

public int getX() {

return x;

}

public void setX(int x) {

this.x = x;

}

public int getY() {

return y;

}

public int getWidth() {

return image.getWidth(null);

}

public int getHeight() {

return image.getHeight(null);

}

public Image getImage() {

return image;

}

public boolean canShoot() {

return canShoot;

}

public void setCanShoot(boolean canShoot) {

this.canShoot = canShoot;

}

public int getDirection() {

return direction;

}

public void reverseDirection() {

direction = -direction;

}

}

class BigEnemy {

private int x;

private int y;

private int direction; // Direction of horizontal movement (left or right)

private boolean canShoot = true;

private final Image image;

public BigEnemy(int x, int y, boolean initialDirection) {

this.x = x;

this.y = y;

this.direction = initialDirection ? 1 : -1;

image = new ImageIcon("boss1.gif").getImage();

}

public void move() {

y += 1;

if (y > 600) {

y = 0;

x = (int) (Math.random() \* 500);

}

}

public int getX() {

return x;

}

public void setX(int x) {

this.x = x;

}

public int getY() {

return y;

}

public int getWidth() {

return image.getWidth(null);

}

public int getHeight() {

return image.getHeight(null);

}

public Image getImage() {

return image;

}

public boolean canShoot() {

return canShoot;

}

public void setCanShoot(boolean canShoot) {

this.canShoot = canShoot;

}

public int getDirection() {

return direction;

}

public void reverseDirection() {

direction = -direction;

}

}

class EnemyBullet {

private double x;

private double y;

private boolean active = true;

public EnemyBullet(double x, double y) {

this.x = x;

this.y = y;

}

public void move() {

y += 2;

if (y > 600) {

active = false;

}

}

public double getX() {

return x;

}

public double getY() {

return y;

}

public boolean isActive() {

return active;

}

public void setActive(boolean active) {

this.active = active;

}

}