Java Game Engine Documentation

Graphical user interface, application

Description automatically generated

Figure 1: Game made using game engine.

(<https://github.com/esaini1/Game-Engine.git>)

**Goal:**

Using the ADT developed in class, build a game engine using Java. The engine should have some sort of tool for managing the background, timing, sprites, motion, collision detection and boundary detection.

**Game Engine Description:**

The following game engine uses Java and builds on the foundation of a game engine that was worked on in class. The game engine has the tools needed for managing the background, timing, sprites, motion, collision detection and boundary detection using two java files. The java files are titled “scene.java” and “sprite.java”. The engine can also run a demo game using the game.java file.

**Dependencies:**

1. **Scene**

The game engine will allow for the declaration of important values to the game engine. This includes “Tile Length” which determines the size of a position on the canvas. The scene length determines the height of the canvas JFrame window. The timer function and sprites can be declared here.

private final int DELAY = 25;

public static final int Tile\_Length = 50;

public static final int Scene\_Length = 8;

public static final int EnemyObjects = 5;

private Timer timer;

private Sprite player;

private ArrayList<Game> enemies;

The JFrame canvas is set to create a window frame and close once complete. Setting pertaining the window can be set including resize and visible. “Scene canvas = new scene()” will initialize the game loop.

class Play {

private static void initWindow() {

JFrame window = new JFrame("ANNIHILATE THE PAWNS!!!");

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

Scene canvas = new Scene();

window.add(canvas);

window.addKeyListener(canvas);

window.setResizable(false);

window.pack();

window.setLocationRelativeTo(null);

window.setVisible(true);

}

public static void main(String[] args) {

SwingUtilities.invokeLater(new Runnable() {

public void run() {

initWindow();

}

});

}

}

public Scene() {

setPreferredSize(new Dimension(Tile\_Length \* Scene\_Length, Tile\_Length \* Scene\_Length));

setBackground(new Color(232, 232, 232));

player = new Sprite();

enemies = populateenemies();

timer = new Timer(DELAY, this);

timer.start();

}

1. **Timers**

The timer is located in the scene function as seen above and will call the actionPerformed() method at every interval as determined earlier by the “DELAY” variable. The actionPerformed() method can be seen below and calls on boundaries to be checked, collisions to be checked and to repaint the canvas at every interval.

public void actionPerformed(ActionEvent e) {

player.boundry();

collision();

repaint();

}

1. **Managing Background**

The draw background function can set the color of the background. The function also gives the ability to create a checkered background to view the possible positions of sprites.

private void drawBackground(Graphics g) {

g.setColor(new Color(214, 214, 214));

for (int row = 0; row < Scene\_Length; row++) {

for (int col = 0; col < Scene\_Length; col++) {

if ((row + col) % 2 == 1) {

g.fillRect(

col \* Tile\_Length,

row \* Tile\_Length,

Tile\_Length,

Tile\_Length

);

}

}

}

}

1. **Collision Detection**

Collision detection is determined by the position of the sprite. The collision action is called at every interval by the previous “actionPerformed()” method. The collision action determines if an enemy has been touched and removes them. The collision detection can also be used for score keeping given the number of touched enemies.

private void collision() {

ArrayList<Game> touchEnemy = new ArrayList<>();

for (Game enemy : enemies) {

if (player.getPos().equals(enemy.getPos())) {

player.addScore(100);

touchEnemy.add(enemy);

}

}

enemies.removeAll(touchEnemy);

}

1. **Sprite**

The sprite can is tracked on the canvas using the “pos” point variable for position of the player and a score is kept for the sprite. The sprite image is then loaded and initialized. The load image method makes it easy to add an image for the sprite using the file name or file path.

public class Sprite {

private BufferedImage image;

private Point pos;

private int score;

public Sprite() {

loadImage();

pos = new Point(0, 0);

score = 0;

}

private void loadImage() {

try {

image = ImageIO.read(new File("images/queen.png"));

} catch (IOException exc) {

System.out.println("Error opening image file: " + exc.getMessage());

}

}

1. **Movement**

Movement of the sprites are performed using arrow keys or collision detection. Every keyboard key has a certain code. When that value is received, it can be used to move the sprite at every interval to given coordinate on the canvas.

@Override

public void keyPressed(KeyEvent e) {

player.keyPressed(e);

}

public void keyPressed(KeyEvent e) {

int key = e.getKeyCode();

if (key == KeyEvent.VK\_UP) {

pos.translate(0, -1);

}

if (key == KeyEvent.VK\_RIGHT) {

pos.translate(1, 0);

}

if (key == KeyEvent.VK\_DOWN) {

pos.translate(0, 1);

}

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

pos.translate(-1, 0);

}

}

1. **Boundary Detection**

The boundary detection accounts for both horizontal and vertical boundaries to ensure that a sprite doesn’t have the ability to leave the canvas. The boundary detection is determined by the window length and position of the sprite.

public void boundry() {

if (pos.x < 0) {

pos.x = 0;

} else if (pos.x >= Scene.Scene\_Length) {

pos.x = Scene.Scene\_Length - 1;

}

if (pos.y < 0) {

pos.y = 0;

} else if (pos.y >= Scene.Scene\_Length) {

pos.y = Scene.Scene\_Length - 1;

}

}

**Demo Game:**Graphical user interface, application

Description automatically generatedA picture containing graphical user interface

Description automatically generated

Figure 2: Demo game in action.

The demo game is nothing to write home about. It a simple game in which the sprite (queen chess piece) can move around the canvas and destroy enemies (pawns). The demo works as shown in figure 2. With each load of the canvas, a new random order of enemies are spawned. Although score detection does exist as mentioned earlier, it is not utilized in this demo. The demo along with the game engine files can be found on GitHub at <https://github.com/esaini1/Game-Engine.git>.

public class Game {

private BufferedImage image;

private Point pos;

public Game(int x, int y) {

loadImage();

pos = new Point(x, y);

}

private void loadImage() {

try {

image = ImageIO.read(new File("images/pawn.png"));

} catch (IOException exc) {

System.out.println("Error opening image file: " + exc.getMessage());

}

}

public void draw(Graphics g, ImageObserver observer) {

g.drawImage(

image,

pos.x \* Scene.Tile\_Length,

pos.y \* Scene.Tile\_Length,

observer

);

}

public Point getPos() {

return pos;

}

**Lessons Learned:**

Adjusting to the Java environment was certainly a challenge as I’ve never actually used it before. However, the documentation I found regarding the use of a canvas were much clearer than JavaScript in my view. I had to learn about the unique syntax of java and all the burdens that come with it. After lots of trial and error, overtime the process became easier and I began to see the similarities and differences compared to previous languages I’ve used in the past.