Simple Shooting Game Project Report

Ishiat Al Mahmood

CSE 078 08419

Supervised by: Tamjid Rahman

Assistant Professor

Department of Computer Science and Engineering

STAMFORD UNIVERSITY BANGLADESH

November 2023

Table of Contents

Introduction

This project aims to create the base-line for a 2D shooting game using the java programming language. The core mechanics of a shooting game which includes basic player movement and mouse tracking are the most important aspect of modern shooting games as opposed to retro games that used directional keys to aim the player. Adding functionality that uses the mouse allows a lot more precision and control to the player and makes the experience more enjoyable

Objective

The core objective of this project is to have a better understanding of a program that needs to run synchronized with all the other methods involved. For this purpose a handler was added that handles all the various methods run in a synchronized manner and achieve their purposes without any hiccups or lag in the game while it’s being played.

Chapter-1: Game Code

src/main/window.java

```
package main;
import java.awt.Canvas;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.image.BufferStrategy;
import javax.swing.JFrame;
import entities.Vector2D;
import graphics.Assets;
import graphics.Camera;
import input.KeyManager;
import input.MouseManager;
import states.GameState;
import states.State;
public class Window implements Runnable {
	public static final int WIDTH = 1000, HEIGHT = 700;
	private final String title = "Game";
	private JFrame window;
	private Canvas canvas;
	private Thread thread;
	private boolean running = false;
	private Graphics g;
	private BufferStrategy bs;
	private GameState gameState;
	private MouseManager mouseManager;
	private KeyManager keyManager;
	private Camera camera;
	private Handler handler;
	private long averageFPS = 0;
	public Window() {
		window = new JFrame(title);
		window.setSize(WIDTH, HEIGHT);
		window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		window.setLocationRelativeTo(null);
		window.setResizable(false);
		window.setVisible(true);
		canvas = new Canvas();
		canvas.setPreferredSize(new Dimension(WIDTH, HEIGHT));
		canvas.setMaximumSize(new Dimension(WIDTH, HEIGHT));
		canvas.setMinimumSize(new Dimension(WIDTH, HEIGHT));
		canvas.setFocusable(true);
		window.add(canvas);
		window.pack();
		mouseManager = new MouseManager();
		keyManager = new KeyManager();
```

```
		window.addMouseMotionListener(mouseManager);
		window.addMouseListener(mouseManager);
		canvas.addMouseMotionListener(mouseManager);
		canvas.addMouseListener(mouseManager);
		canvas.addKeyListener(keyManager);}
private void init() {
		Assets.init();
		camera = new Camera(new Vector2D());
		handler = new Handler(this, mouseManager, keyManager);
		gameState = new GameState(handler);
		State.currentState = gameState;}
private void update() {
		keyManager.update();
		if (State.currentState != null)
			State.currentState.update();}
Color color = new Color(0, 0, 0, 125);
private void render() {
		bs = canvas.getBufferStrategy();
		if (bs == null) {
			canvas.createBufferStrategy(3);
			return;}
		g = bs.getDrawGraphics();
		if (State.currentState != null)
			State.currentState.render(g);
		g.drawString("FPS: " + averageFPS, 600, 25);
		g.dispose();
		bs.show();}
@Override
public void run() {
		init();
		int fps = 60;
		double timePerTick = 1000000000 / fps;
		double delta = 0;
		long now;
		long lastTime = System.nanoTime();
		long ticks = 0;
		long timer = 0;
		while (running) {
			now = System.nanoTime();
			delta += (now - lastTime) / timePerTick;
			timer += now - lastTime;
			lastTime = now;
			if (delta >= 1) {
				update();
				render();
				delta--;
				ticks++;}
			if (timer >= 1000000000) {
				averageFPS = ticks;
				ticks = 0;
				timer = 0;}}
		stop();}
```

```
	private synchronized void start() {
		if (running)
			return;
		running = true;
		thread = new Thread(this);
		thread.start();}
	private synchronized void stop() {
		if (!running)
			return;
		try {
			thread.join();
			running = false;
		} catch (InterruptedException e) {
			e.printStackTrace();}}
	public Camera getCamera() {
		return camera;}
	public static void main(String[] args) {
		new Window().start();}}
```

Src/main/handler.java

```
package main;

import input.KeyManager;
import input.MouseManager;

public class Handler {
	private MouseManager mouseManager;
	private KeyManager keyManager;
	private Window window;

	public Handler(Window window, MouseManager mouseManager, KeyManager keyManager){
		this.window = window;
		this.mouseManager = mouseManager;
		this.keyManager = keyManager;
	}

	public MouseManager getMouseManager() {
		return mouseManager;
	}

	public void setMouseManager(MouseManager mouseManager) {
		this.mouseManager = mouseManager;
	}

	public KeyManager getKeyManager() {
		return keyManager;
	}

	public void setKeyManager(KeyManager keyManager) {
		this.keyManager = keyManager;
	}
```

```
	public Window getWindow() {
		return window;
	}

	public void setWindow(Window window) {
		this.window = window;
	}


}
```

Src/input/keymanager.java

```
package input;

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

public class KeyManager implements KeyListener{

	public static boolean[] keys;
	public static boolean up,left,right, down, one, two, reload;


	public KeyManager(){
		keys = new boolean[256];
		up = false;
		left = false;
		right = false;
		down = false;
	}
	public void update(){
		up = keys[KeyEvent.VK_W];
		left = keys[KeyEvent.VK_A];
		right = keys[KeyEvent.VK_D];
		down = keys[KeyEvent.VK_S];
		one = keys[KeyEvent.VK_1];
		two = keys[KeyEvent.VK_2];
		reload = keys[KeyEvent.VK_R];
	}


	@Override
	public void keyPressed(KeyEvent e) {
		keys[e.getKeyCode()] = true;
	}

	@Override
	public void keyReleased(KeyEvent e) {
		keys[e.getKeyCode()] = false;
```

```
	}

	@Override
	public void keyTyped(KeyEvent e) {}
}
```

Src/input/mousemanager.java

```
package input;

import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;

import entities.Vector2D;

public class MouseManager implements MouseMotionListener, MouseListener{

	public static Vector2D position;
	public static boolean leftMouseButton = false;
	public static boolean rightMouseButton = false;


	public MouseManager(){
		position = new Vector2D();
	}

	@Override
	public void mouseDragged(MouseEvent e) {
		position.setX(e.getX());
		position.setY(e.getY());
	}


	@Override
	public void mouseMoved(MouseEvent e) {
		position.setX(e.getX());
		position.setY(e.getY());
	}

	@Override
	public void mousePressed(MouseEvent e) {
		if(e.getButton() == MouseEvent.BUTTON1)
			leftMouseButton = true;
		if(e.getButton() == MouseEvent.BUTTON3)
			rightMouseButton = true;
	}

	@Override
	public void mouseReleased(MouseEvent e) {
		if(e.getButton() == MouseEvent.BUTTON1)
			leftMouseButton = false;
```

```
                if(e.getButton() == MouseEvent.BUTTON3)
                        rightMouseButton = false;
        }

        @Override
        public void mouseEntered(MouseEvent e) {}
        @Override
        public void mouseExited(MouseEvent e) {}
        @Override
        public void mouseClicked(MouseEvent e) {}
}
```

Src/entities/creatures/creature.java

```
package entities.creatures;

import java.awt.geom.AffineTransform;

import entities.Entity;
import entities.Vector2D;
import main.Handler;

public abstract class Creature extends Entity{


   protected Vector2D velocity;
   protected double maxVelocity;
   protected AffineTransform at;
   protected double angle;
   protected int health;


   public Creature(Handler handler, Vector2D position, int width, int height, double maxVelocity, int health) {
      super(handler, position, width, height);
      this.velocity = new Vector2D();
      this.maxVelocity = maxVelocity;
      angle = 0;
      this.health = health;
   }

   public Vector2D getVelocity() {
      return velocity;
   }

   public void setVelocity(Vector2D velocity) {
      this.velocity = velocity;
   }

   public double getMaxVelocity() {
      return maxVelocity;
   }
```

```
    public void setMaxVelocity(double maxVelocity) {
        this.maxVelocity = maxVelocity;
    }

    public int getHealth() {
        return health;
    }

    public void setHealth(int health) {
        this.health = health;
    }
}
```

Src/entity/creatrues/player.java

```
package entities.creatures;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.geom.AffineTransform;

import entities.Bullet;
import entities.Entity;
import entities.Gun;
import entities.Vector2D;
import graphics.Animation;
import graphics.Assets;
import graphics.Sound;
import input.KeyManager;
import input.MouseManager;
import tiles.World;

public class Player extends Creature{

    public static final int PLAYERSIZE = 112;
    public static final int MAXHEALTH = 400;

    // current gun

    private static Gun currentGun;

    // pistol animations

    private Animation pistolIdle, pistolReload, pistolShoot;

    // rifle animations

    private Animation rifleIdle, rifleReload, rifleShoot;
```

```
// guns

private Gun pistol, rifle;

// sounds

private Sound rifleShootSound, rifleReloadSound;
private Sound pistolShootSound, pistolReloadSound;

// others

private Vector2D shootPoint;
private Vector2D creatureCenter;
private Vector2D newPoint;
private double xMove, yMove;
private World world;


public Player(Vector2D position, double maxVelocity, World world) {
   super(world.getHandler(), position, PLAYERSIZE - 40, PLAYERSIZE - 40, maxVelocity, MAXHEALTH);
   this.world = world;
   this.velocity = new Vector2D(maxVelocity, maxVelocity);
   pistolIdle = new Animation(Assets.pistolIdle, 20);
   pistolReload = new Animation(Assets.pistolReload, 100);
   pistolShoot = new Animation(Assets.pistolShootAnim, 80);
   pistolShootSound = new Sound(Assets.pistolShoot);
   pistolShootSound.changeVolume(-10);
   pistolReloadSound = new Sound(Assets.pistolReloadSound);

   rifleIdle = new Animation(Assets.rifleIdle, 20);
   rifleReload = new Animation(Assets.rifleReload, 100);
   rifleShoot = new Animation(Assets.rifleShootAnim, 80);
   rifleShootSound = new Sound(Assets.rifleShoot);
   rifleShootSound.changeVolume(-5);
   rifleReloadSound = new Sound(Assets.rifleReloadSound);

   shootPoint = new Vector2D();

   pistol = new Gun(Assets.pistolSkin, pistolIdle, pistolReload, pistolShoot,pistolShootSound,
         pistolReloadSound, this, 700,
         9, 2700);
   rifle = new Gun(Assets.ak47, rifleIdle, rifleReload, rifleShoot, rifleShootSound,
         rifleReloadSound, this, 100,
         30, 9000);

   currentGun = pistol;

}

@Override
public void update() {
```

```
int x = (int) handler.getWindow().getCamera().getOffset().getX();
int y = (int) handler.getWindow().getCamera().getOffset().getY();


bounds.x = (int)(position.getX() - x + 15);
bounds.y = (int)(position.getY() - y + 15);

xMove = 0;
yMove = 0;


if(KeyManager.up)
   yMove = - velocity.getY();
if(KeyManager.left)
   xMove = -velocity.getX();
if(KeyManager.right)
   xMove = velocity.getX();
if(KeyManager.down)
   yMove = velocity.getY();
if(KeyManager.one)
   currentGun = pistol;
if(KeyManager.two)
   currentGun = rifle;
if(KeyManager.reload && currentGun.getRound() != currentGun.getBulletsPerRound() &&
      currentGun.getTotalBullets() > 0)
   currentGun.reload();

double width = MouseManager.position.getX() - (position.getX()  - x + PLAYERSIZE /2);
double height = MouseManager.position.getY() - (position.getY() - y + PLAYERSIZE / 2);

angle = Math.atan(height / width);

if(width < 0)
   angle = -Math.PI + angle;

creatureCenter = new Vector2D(shootPoint.getX() - (position.getX() + PLAYERSIZE/2),
      shootPoint.getY() - (position.getY() + PLAYERSIZE/2));

newPoint = new Vector2D(creatureCenter.getX()*Math.cos(angle) - creatureCenter.getY()*Math.sin(angle),
      creatureCenter.getX()*Math.sin(angle) + creatureCenter.getY()*Math.cos(angle));

shootPoint.setX(newPoint.getX() - x + position.getX() + PLAYERSIZE/2);
shootPoint.setY(newPoint.getY() - y +  position.getY() + PLAYERSIZE/2);

if(MouseManager.leftMouseButton){


   Vector2D bulletDirection = new Vector2D(Math.cos(angle), Math.sin(angle));
   bulletDirection = bulletDirection.normalize();

   Bullet bullet = new Bullet(world.getHandler(), shootPoint, bulletDirection, world);
```

```
            currentGun.shoot(bullet);

        }

        // update  shooting point
        shootPoint = new Vector2D(position.getX() + PLAYERSIZE - 7, position.getY() + PLAYERSIZE - 27);

        getInput();
        handler.getWindow().getCamera().centerOnEntity(this);
        currentGun.update();

    }

    private void getInput(){

        if(collision(0, yMove) == null)
            yMove();
        if(collision(xMove, 0) == null)
            xMove();
    }
    private void yMove(){
        position.setY(position.getY()+ yMove);
    }
    private void xMove(){
        position.setX(position.getX()+ xMove);
    }

    public Entity collision(double xMove, double yMove){
        for(int i = 0; i < world.getObstacles().size(); i++)
        {
            Entity e = world.getObstacles().get(i);

            if(e.getDimensions(0, 0).intersects(this.getDimensions(xMove, yMove)))
                return e;
        }
        return null;

    }


    @Override
    public void render(Graphics g) {

        int x = (int) handler.getWindow().getCamera().getOffset().getX();
        int y = (int) handler.getWindow().getCamera().getOffset().getY();

        at = AffineTransform.getTranslateInstance(position.getX() - x, position.getY() - y);


        at.rotate(angle, PLAYERSIZE/2, PLAYERSIZE/2);

        Graphics2D g2d = (Graphics2D)g;
```

```
        g2d.drawImage(currentGun.getCurrentAnimation().getCurrentFrame(), at, null);

        if(health > 300)
            g2d.setColor(Color.GREEN);
        else if(health > 200)
            g2d.setColor(Color.YELLOW);
        else if(health > 100)
            g2d.setColor(Color.ORANGE);
        else if(health > 50)
            g2d.setColor(Color.RED);
        else if(health > 0)
            g2d.setColor(Color.BLACK);

        g2d.fillRect(50, 25, health, 25);

    }

    public void hit(){
        health -= 20;
    }

    public World getWorld(){
        return world;
    }

    public Gun getCurrentGun(){
        return currentGun;
    }
}
```

Src/entity/creatures/zombie.java

```
package entities.creatures;

import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Rectangle;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.geom.AffineTransform;

import javax.swing.Timer;

import blood.Blood;
import entities.Vector2D;
import graphics.Animation;
import graphics.Assets;
import graphics.Sound;

public class Zombie extends Creature{
```

```
public static final int ZOMBIESIZE = 128;
public static final int MAXHEALTH = 100;

private double mass;
private Steering steering;
private Player player;
private Vector2D acceleration, heading;
private double maxForce;

private Animation walkAnimation, attackAnimation;

private Animation currentAnimation;

private Timer attackDelay;

private boolean attacking = false;

private double distanceToPlayer = 0;

private Sound zombieBite;

public Zombie(Vector2D position, double maxVelocity, Player player) {
   super(player.getWorld().getHandler(), position, ZOMBIESIZE, ZOMBIESIZE, maxVelocity, MAXHEALTH);
   this.player = player;
   zombieBite = new Sound(Assets.zombieBite);
   steering = new Steering(this, player, handler);
   acceleration = new Vector2D();
   heading = new Vector2D();
   mass = 5;
   maxForce = 5;

   attackDelay = new Timer(350, new ActionListener(){

      @Override
      public void actionPerformed(ActionEvent e) {
         if(distanceToPlayer < Player.PLAYERSIZE)
         {
            zombieBite.playSound();
            player.hit();
         }

         attackDelay.stop();
         attacking = false;
         currentAnimation = walkAnimation;
      }

   });
   walkAnimation = new Animation(Assets.zombie, 20);
   attackAnimation = new Animation(Assets.zombieAttack, 35);
   currentAnimation = walkAnimation;
```

```
}

@Override
public void update() {
    int x = (int) handler.getWindow().getCamera().getOffset().getX();
    int y = (int) handler.getWindow().getCamera().getOffset().getY();

    bounds = new Rectangle((int)position.getX() - x + 25, (int)position.getY() - y + 25,
            ZOMBIESIZE - 50, ZOMBIESIZE - 50);

    Vector2D seekForce = steering.seek(new Vector2D(player.getPosition().getX(), player.getPosition().getY()));
    Vector2D avoidanceForce = steering.obstacleAvoidance();
    Vector2D separationForce = steering.separation();

    Vector2D steeringForce = new Vector2D();



    if(!avoidanceForce.isZero())
        seekForce = new Vector2D();


    steeringForce.setX(seekForce.getX() + avoidanceForce.getX() + separationForce.getX());
    steeringForce.setY(seekForce.getY() + avoidanceForce.getY() + separationForce.getY());

    steeringForce.divideByScalar(10);

    steeringForce.truncate(maxForce);

    acceleration.setX(acceleration.getX() + steeringForce.getX());
    acceleration.setY(acceleration.getY() + steeringForce.getY());

    acceleration.divideByScalar(mass);

    velocity.setX(velocity.getX() + acceleration.getX());
    velocity.setY(velocity.getY() + acceleration.getY());

    velocity.truncate(maxVelocity);


    Vector2D toPlayer = player.getPosition().substract(position);
    distanceToPlayer = toPlayer.getMagnitude();


    if(distanceToPlayer < Player.PLAYERSIZE/2 && !attackDelay.isRunning())
    {
        attacking = true;
        attackDelay.start();
        currentAnimation = attackAnimation;
        currentAnimation.setIndex();
```

```
    }

    if(attacking)
        velocity.zero();


    position.setX(position.getX() + velocity.getX());
    position.setY(position.getY() + velocity.getY());

    if(velocity.getMagnitude() > 0){
        heading = velocity.normalize();

        angle = Math.acos(heading.dot(new Vector2D(1,0)));
        if(heading.getY() < 0)
            angle *= -1;
    }

    currentAnimation.update();
}

@Override
public void render(Graphics g) {

    int x = (int) handler.getWindow().getCamera().getOffset().getX();
    int y = (int) handler.getWindow().getCamera().getOffset().getY();

    at = AffineTransform.getTranslateInstance(position.getX() - x,
            position.getY() - y);

    at.rotate(angle, ZOMBIESIZE/2, ZOMBIESIZE/2);

    Graphics2D g2d = (Graphics2D)g;

    g2d.drawImage(currentAnimation.getCurrentFrame(), at, null);

    //steering.render(g2d);
}

public Vector2D getHeading(){
    return heading;
}

public double getMaxForce(){
    return maxForce;
}
public double getRadius(){
    return bounds.width/2;
}

public Vector2D getCenter(){
    return new Vector2D(bounds.x + bounds.width/2, bounds.y + bounds.height/2);
}
```

```
    public void hit(){
        health -= 20;
        if(health <= 0){
            player.getWorld().getBloodSplats().add(new Blood(position, player.getWorld()));
            player.getWorld().getZombiesAndBullets().remove(this);
        }

    }
}
```

Src/entity/entity.java

```
package entities;
import java.awt.Graphics;
import java.awt.Rectangle;

import main.Handler;

public abstract class Entity {

    protected Vector2D position;
    protected int width, height;
    protected Rectangle bounds;
    protected Handler handler;

    public Entity(Handler handler, Vector2D position, int width, int height){
        this.position = position;
        this.width = width;
        this.height = height;

        bounds = new Rectangle((int)position.getX(), (int)position.getY(), width, height);
        this.handler = handler;

    }

    public abstract void update();

    public abstract void render(Graphics g);

    public Vector2D getPosition(){
        return position;
    }

    public Rectangle getBounds(){
        return bounds;
    }

    public Rectangle getDimensions(double xMove, double yMove){
        return new Rectangle((int)(bounds.x + xMove), (int)(bounds.y + yMove), bounds.width, bounds.height);
    }
```

```
    public int getWidth() {
        return width;
    }

    public int getHeight() {
        return height;
    }


}
```

Src/entity/bullet.java

```
package entities;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Point;

import entities.creatures.Zombie;
import graphics.Assets;
import graphics.Sound;
import main.Handler;
import main.Window;
import particles.Particle;
import tiles.Tile;
import tiles.World;

public class Bullet extends Entity{

    private final Vector2D velocity;
    private final static int BULLETDIAMETER = 6;
    private final int bulletVelocity = 20;
    private Sound zombieHit;
    private World world;


    public Bullet(Handler handler, Vector2D position, Vector2D velocity, World world) {
        super(handler, position, BULLETDIAMETER, BULLETDIAMETER);
        this.velocity = velocity;
        zombieHit = new Sound(Assets.zombieHit);
        this.world = world;
    }

    @Override
    public void update() {

        position.setX(position.getX() + (velocity.getX()*bulletVelocity));
        position.setY(position.getY() + (velocity.getY()*bulletVelocity));
```

```
    Point center = new Point((int)position.getX() - BULLETDIAMETER/2, (int)position.getY() -
BULLETDIAMETER/2);


    for(int i = 0; i < world.getZombiesAndBullets().size(); i++)
    {

       if(world.getZombiesAndBullets().get(i) instanceof Bullet)
          continue;

       Zombie zombie = (Zombie)world.getZombiesAndBullets().get(i);

          if(zombie.getBounds().contains(center))
             {
             zombieHit.playSound();
                for(int j = 0; j <= 360; j+=15)
                {

                   int speed = (int)(Math.random()*6);
                   int life  = (int)(Math.random()*20);
                   int size = (int)(Math.random()*8);

                   world.getParticles().add(new Particle(new Vector2D(position.getX(), position.getY()),
                         size, life, speed, j, Color.RED));
                }
                zombie.hit();
                world.getZombiesAndBullets().remove(this);
             }
    }

    for(int i = 0; i < world.getObstacles().size(); i++)
    {
       Entity o = world.getObstacles().get(i);

       if(o.getBounds().contains(center))
       {
          for(int j = 0; j <= 360; j+=15)
          {

             int speed = (int)(Math.random()*6);
             int life  = (int)(Math.random()*30);
             int size = (int)(Math.random()*10);

             world.getParticles().add(new Particle(new Vector2D(position.getX(), position.getY()),
                   size, life, speed, j, new Color(95, 45, 0)));
          }
          world.getZombiesAndBullets().remove(this);
       }
    }
    if(position.getX() > World.WIDTH*Tile.TILESIZE || position.getX() < 0)
       world.getZombiesAndBullets().remove(this);
    if(position.getY() > World.HEIGHT*Tile.TILESIZE || position.getY() < 0)
```

```
            world.getZombiesAndBullets().remove(this);


    }

    @Override
    public void render(Graphics g) {
        g.setColor(Color.BLACK);

        g.fillOval((int)(position.getX() - BULLETDIAMETER/2),
                (int)(position.getY() - BULLETDIAMETER/2),
                BULLETDIAMETER, BULLETDIAMETER);
    }
}
```

Src/entity/tree.java

```
package entities;

import java.awt.Graphics;
import java.awt.image.BufferedImage;

import main.Handler;

public class Tree extends Entity{

    private BufferedImage texture;
    private int cameraXoffset, cameraYoffset;

    public Tree(Handler handler, Vector2D position, BufferedImage texture) {
        super(handler, position, texture.getWidth(), texture.getHeight());
        this.texture = texture;

        bounds.x = bounds.x + 25 ;
        bounds.y = bounds.y + 20;
        bounds.width = bounds.width - 50;
        bounds.height = bounds.height - 50;


    }

    @Override
    public void update() {
        cameraXoffset = (int) handler.getWindow().getCamera().getOffset().getX();
        cameraYoffset = (int) handler.getWindow().getCamera().getOffset().getY();

        bounds.x = (int)position.getX() + 25 - cameraXoffset;
        bounds.y = (int)position.getY() + 20 - cameraYoffset;
    }

    @Override
```

```
    public void render(Graphics g) {
        g.drawImage(texture, (int)position.getX() - cameraXoffset, (int)position.getY() - cameraYoffset, null);
    }

    public Vector2D getCenter(){
        return new Vector2D(position.getX() + width/2, position.getY() + height/2);
    }

    public double getRadius(){
        return width/2;
    }
}
```

Src/entity/vector2d.java

```
package entities;

public class Vector2D {
    private double x, y;

    public Vector2D(double x, double y){
        this.x = x;
        this.y = y;
    }

    public Vector2D(){
        x = 0;
        y = 0;
    }

    public void zero(){
        x = 0;
        y = 0;
    }


    public double dot(Vector2D v){
        return x*v.getX() + y*v.getY();
    }

    public boolean isZero(){
        if(x == 0 && y == 0)
            return true;
        return false;
    }

    public static double Distance(Vector2D a, Vector2D b){

        return a.substract(b).getMagnitude();
    }
```

```
public Vector2D normalize(){
   return new Vector2D(x / getMagnitude(), y / getMagnitude());
}

public Vector2D substract(Vector2D v){
   return new Vector2D(x - v.getX(), y - v.getY());
}

public Vector2D add(Vector2D v){
   return new Vector2D(x + v.getX(), y + v.getY());
}

public double getMagnitude(){
   return Math.sqrt(x*x + y*y);
}


public Vector2D multyplyByScalar(double scalar){
   x *= scalar;
   y *= scalar;
   return this;
}
public Vector2D divideByScalar(double scalar){
   x /= scalar;
   y /= scalar;
   return this;
}

public void truncate(double value){
   if(x > value)
      x = value;
   if(y > value )
      y = value;
   if(x < -value)
      x = -value;
   if(y < -value )
      y = -value;
}

public double getX() {
   return x;
}

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

public double getY() {
   return y;
}

public void setY(double y) {
```

```
        this.y = y;
    }

    public String toString(){
        return "x: "+(int)x+" y: "+(int)y;
    }
}
```

Src/particle/particle.java

```
package particles;

import java.awt.Color;
import java.awt.Graphics;

import entities.Vector2D;

public class Particle {
    private Vector2D position;
    private int size, life;
    private int speed;
    private int alpha = 130;
    private final Color color;
    private Vector2D direction;

    public Particle(Vector2D position, int size, int life, int speed, double angle, Color color){
        this.color = new Color(color.getRed(), color.getGreen(), color.getBlue(), alpha);
        this.size = size;
        this.life = life;
        this.speed = speed;
        angle = Math.toRadians(angle);
        direction = new Vector2D(Math.cos(angle), Math.sin(angle));
        direction.normalize();
        this.position = position;
    }


    public boolean update(){
        position.setX(position.getX() + direction.getX()*speed);
        position.setY(position.getY() + direction.getY()*speed);
        life --;
        if(life <= 0)
            return true;

        return false;

    }

    public void render(Graphics g){
        g.setColor(color);
        g.fillRect((int)position.getX() - size/2, (int)position.getY() - size/2, size, size);
```

```
    }


}
```

Src/blood/blood.java

```
package blood;
import java.awt.AlphaComposite;
import java.awt.Graphics;
import java.awt.Graphics2D;

import entities.Vector2D;
import graphics.Assets;
import main.Handler;
import tiles.World;

public class Blood {

    private Vector2D position;
    private float alpha;
    private World world;

    public Blood(Vector2D position, World world){
        this.position = position;
        this.world = world;
        alpha = 1;
    }

    public void render(Graphics g){

        int x = (int) world.getHandler().getWindow().getCamera().getOffset().getX();
        int y = (int) world.getHandler().getWindow().getCamera().getOffset().getY();

        Graphics2D g2d = (Graphics2D)g;

        g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER, alpha));

        g2d.drawImage(Assets.blood1, (int)(position.getX() - x), (int)(position.getY() - y), null);

        g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER, 1));

        alpha -= 0.001f;
        if(alpha < 0)
            world.getBloodSplats().remove(this);
    }

}
```

Src/graphics/animation.java

```
package graphics;

import java.awt.image.BufferedImage;

public class Animation {

    private BufferedImage[] frames;
    private int velocity, index;

    private long time, lastTime;



    public Animation(BufferedImage[] frames, int velocity){
        this.frames = frames;
        this.velocity = velocity;

        time = 0;
        lastTime = 0;
        index = 0;

    }

    public void update(){

        time += System.currentTimeMillis() - lastTime;

        lastTime = System.currentTimeMillis();

        if(time > velocity){
            time = 0;
            index ++;
            if(index == frames.length){
                index = 0;
            }

        }

    }

    public BufferedImage getCurrentFrame(){
        return frames[index];
    }

    public void setIndex(){
        index = 0;
    }

    public int getIndex(){
        return index;
    }
```

```
    public int getLenght(){
        return frames.length;
    }

    public int getVelocity(){
        return velocity;
    }
}
```

Src/graphics/assets.java

```
package graphics;

import java.awt.image.BufferedImage;

import javax.sound.sampled.Clip;

import utilities.Utilities;

public class Assets {

    // sprite sheets
    public static SpriteSheet mountain = new SpriteSheet(Utilities.loadImage("/mountain/mountain.png"));


    public static BufferedImage[] pistolIdle = new BufferedImage[20];
    public static BufferedImage[] rifleIdle = new BufferedImage[20];

    public static BufferedImage[] pistolReload = new BufferedImage[15];
    public static BufferedImage[] rifleReload = new BufferedImage[20];

    public static BufferedImage[] pistolShootAnim = new BufferedImage[3];
    public static BufferedImage[] rifleShootAnim = new BufferedImage[3];

    public static BufferedImage[] zombie = new BufferedImage[17];
    public static BufferedImage[] zombieAttack = new BufferedImage[9];

    // blood splats

    public static BufferedImage blood1;

    // guns skin

    public static BufferedImage pistolSkin, ak47, rifleLoader;


    //sounds

    public static Clip pistolShoot, rifleShoot, background, zombieHit, pistolReloadSound,
    rifleReloadSound,emptyGun, zombieBite;
```

```
//tiles

public static BufferedImage grass, dirt;

//objects

public static BufferedImage tree;




//trees
public static BufferedImage[] trees = new BufferedImage[4];


public static void init(){

    // animations

    for(int i = 0; i<pistolIdle.length; i++)
        pistolIdle[i] = Utilities.loadImage("/player/idle/pistolIdle/"+i+".png");
    for(int i = 0; i<rifleIdle.length; i++)
        rifleIdle[i] = Utilities.loadImage("/player/idle/rifleIdle/"+i+".png");
    for(int i = 0; i<pistolReload.length; i++)
        pistolReload[i] = Utilities.loadImage("/player/reload/pistol/"+i+".png");
    for(int i = 0; i<rifleReload.length; i++)
        rifleReload[i] = Utilities.loadImage("/player/reload/rifle/"+i+".png");
    for(int i = 0; i<pistolShootAnim.length; i++)
        pistolShootAnim[i] = Utilities.loadImage("/player/shoot/pistol/"+i+".png");
    for(int i = 0; i<rifleShootAnim.length; i++)
        rifleShootAnim[i] = Utilities.loadImage("/player/shoot/rifle/"+i+".png");
    for(int i = 0; i<zombie.length; i++)
        zombie[i] = Utilities.loadImage("/zombie/walk/"+i+".png");
    for(int i = 0; i<zombieAttack.length; i++)
        zombieAttack[i] = Utilities.loadImage("/zombie/attack/"+i+".png");
    for(int i = 0; i < trees.length; i++)
        trees[i] = Utilities.loadImage("/tress/tree"+i+".png");
    tree = Utilities.loadImage("/obstacles/tree.png");
    grass = Utilities.loadImage("/tiles/grass.png");
    dirt = Utilities.loadImage("/tiles/dirt.png");
    pistolSkin = Utilities.loadImage("/guns/pistol.png");
    ak47 = Utilities.loadImage("/guns/ak-47.png");
    rifleLoader = Utilities.loadImage("/guns/rifleLoader.png");
    blood1 = Utilities.loadImage("/zombie/blood/bloodsplat.png");
    pistolShoot = Utilities.LoadSound("/pistol.wav");
    rifleShoot = Utilities.LoadSound("/machineGun.wav");
    background = Utilities.LoadSound("/background.wav");
    zombieHit = Utilities.LoadSound("/zombiehit.wav");
    pistolReloadSound = Utilities.LoadSound("/pistolreload.wav");
    rifleReloadSound = Utilities.LoadSound("/riflereload.wav");
    emptyGun = Utilities.LoadSound("/emptygun.wav");
```

```
        zombieBite = Utilities.LoadSound("/zombieBite.wav");    }}
```

Src/graphics/camera.java

```
package graphics;
import entities.Entity;
import entities.Vector2D;
import main.Window;
import tiles.Tile;
import tiles.World;
public class Camera {
    private Vector2D offSet;
    public Camera(Vector2D offSet){
        this.offSet = offSet;
    }
    public void centerOnEntity(Entity e){
        offSet.setX(e.getPosition().getX() - Window.WIDTH/2 + e.getWidth()/2);
        offSet.setY(e.getPosition().getY() - Window.HEIGHT/2 + e.getHeight()/2);
        checkBlankSpace();
    }
    public void checkBlankSpace(){
        if(offSet.getX() < 0)
            offSet.setX(0);
        if(offSet.getY() < 0)
            offSet.setY(0);
        if(offSet.getX() > World.WIDTH*Tile.TILESIZE - Window.WIDTH)
            offSet.setX(World.WIDTH*Tile.TILESIZE - Window.WIDTH);
        if(offSet.getY() > World.HEIGHT*Tile.TILESIZE - Window.HEIGHT)
            offSet.setY(World.HEIGHT*Tile.TILESIZE - Window.HEIGHT);
    }
    public void move(Vector2D v){
        offSet.setX(offSet.getX() +  v.getX());
        offSet.setY(offSet.getY() +  v.getY());
    }
    public Vector2D getOffset(){
        return offSet;
    }
    public void setOffSet(Vector2D offSet){
        this.offSet = offSet;
    }
}
```

Src/graphics/sound.java

```
package graphics;
import javax.sound.sampled.Clip;
import javax.sound.sampled.FloatControl;
public class Sound {
    private Clip sound;
    private FloatControl volume;
    public Sound(Clip sound){
```

```
        this.sound = sound;
        volume =  (FloatControl) sound.getControl(FloatControl.Type.MASTER_GAIN);
    }
    public void playSound(){
        sound.setFramePosition(0);
        sound.start();
    }
    public void loopSound(){
        sound.loop(Clip.LOOP_CONTINUOUSLY);
    }
    public void stopSound(){
        sound.stop();
        try {
            Thread.sleep(10);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    public int getFramePosition(){
        return sound.getFramePosition();
    }
    public int getFrameLenght(){
        return sound.getFrameLength();
    }
    public void changeVolume(float value){
        volume.setValue(value);
    }
    public boolean isRunning(){
        return sound.isRunning();
    }
}
```

Src/states/gamestate.java

```
package states;
import java.awt.Graphics;
import graphics.Assets;
import graphics.Sound;
import main.Handler;
import tiles.World;
public class GameState extends State{
    private Sound background;
    private World world;
    public GameState(Handler handler){
        super(handler);
        background = new Sound(Assets.background);
        background.loopSound();
        world = new World(handler);}
    public void update() {
        world.update();}
    public void render(Graphics g) {
```

```
        world.render(g);}    }
```

src/states/state.java

```
package states;
import java.awt.Graphics;
import main.Handler;
public abstract class State {
    public static State currentState = null;
    protected Handler handler;
    public State(Handler handler){
        this.handler = handler;
    }
       public abstract void update();
    public abstract void render(Graphics g);
    }
```

Src/tiles/world.java

```
package tiles;
import java.awt.Graphics;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import javax.swing.Timer;
import blood.Blood;
import entities.Entity;
import entities.Tree;
import entities.Vector2D;
import entities.creatures.Player;
import entities.creatures.Zombie;
import graphics.Assets;
import main.Handler;
import main.Window;
import particles.Particle;
public class World {
    public static int WIDTH;
    public static int HEIGHT;
    private int[][] tiles = new int[][]{
        {0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0},
        {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
        {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
        {0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0},
        {0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0},
        {0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0},
        {0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1},
```

```
    {0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1},
};
private Handler handler;
private ArrayList<Entity> zombiesAndBullets;
private ArrayList<Particle> particles;
private ArrayList<Entity> obstacles;
private ArrayList<Blood> bloodSplats;
private Player player;
private Timer timer = new Timer(2000, new ActionListener()
{
    public void actionPerformed(ActionEvent e) {
        int x = (int)(Math.random()*1200); // use these to spawn zombies at random position
        int y = (int)(Math.random()*1000);
        zombiesAndBullets.add(new Zombie(new Vector2D(1000, 1000), 2, player));
    }
});
   public World(Handler handler){
    this.handler = handler;
    WIDTH = tiles[0].length;
    HEIGHT = tiles.length;
    zombiesAndBullets = new ArrayList<Entity>();
    particles = new ArrayList<Particle>();
    obstacles = new ArrayList<Entity>();
    bloodSplats = new ArrayList<Blood>();
    player = new Player(new Vector2D(500, 500), 4, this);
    timer.start();
    obstacles.add(new Tree(handler, new Vector2D(400, 700), Assets.tree));
    obstacles.add(new Tree(handler, new Vector2D(100, 600), Assets.tree));
    obstacles.add(new Tree(handler, new Vector2D(200, 300), Assets.tree));
    obstacles.add(new Tree(handler, new Vector2D(300, 400), Assets.tree));
    obstacles.add(new Tree(handler, new Vector2D(700, 200), Assets.tree));
}
   public void update(){
    player.update();
    for(int i = 0; i < zombiesAndBullets.size(); i++)
        zombiesAndBullets.get(i).update();
    for(int i = 0; i < particles.size(); i++)
        if(particles.get(i).update())
            particles.remove(i);
    for(int i = 0; i < obstacles.size(); i++)
        obstacles.get(i).update();
}
   public void render(Graphics g){
    int x = (int) handler.getWindow().getCamera().getOffset().getX();
    int y = (int) handler.getWindow().getCamera().getOffset().getY();
    int xStart = (int)Math.max(0, x/Tile.TILESIZE);
    int yStart = (int)Math.max(0, y/Tile.TILESIZE);
    int xEnd = (int)Math.min(WIDTH, (x + Window.WIDTH)/Tile.TILESIZE + 1);
    int yEnd = (int)Math.min(HEIGHT, (y + Window.WIDTH)/Tile.TILESIZE + 1);
    for(int i = yStart; i < yEnd; i++){
        for(int j = xStart; j < xEnd; j++){
            getTile(i, j).render(g, new Vector2D(j*Tile.TILESIZE - x,
```

```
                    i*Tile.TILESIZE - y));
            }
        }
        for(int i = 0; i < bloodSplats.size(); i++)
            bloodSplats.get(i).render(g);
        player.render(g);
        for(int i = 0; i < particles.size(); i++)
            particles.get(i).render(g);
        for(int i = 0; i < obstacles.size(); i++)
            obstacles.get(i).render(g);
        for(int i = 0; i < zombiesAndBullets.size(); i++)
            zombiesAndBullets.get(i).render(g);
        player.getCurrentGun().render(g);
    }
    public Tile getTile(int x, int y){
        Tile tile = Tile.tiles[tiles[x][y]];
        return tile;
    }
    public Handler getHandler(){
        return handler;
    }
    public ArrayList<Entity> getZombiesAndBullets(){
        return zombiesAndBullets;
    }
    public ArrayList<Particle> getParticles(){
        return particles;
    }
    public ArrayList<Entity> getObstacles(){
        return obstacles;
    }
    public ArrayList<Blood> getBloodSplats(){
        return bloodSplats;
    }
}
```

Chapter-2: Game Design

Core Mechanics.

The core mechanics of a shooting game involve player movement, enemy movement, and shooting. In the background there are a lot more factors that come into play such as rendering the players/enemies and their respective animations. Tracking the position of the characters and their behavior while simultaneously running everything in sync can prove to be quite a challenge. Here's how I managed to overcome said challenges.

Game loop

The game loop can be considered the heart of the game because much like a heart it has to beat in sync and make sure every other part of the code works as they should. Luckily for me, due to my previous knowledge in video games I was able to procure the game loop made by Notch (The developer of Minecraft) which is regarded as the best game loop handler for java. Using that game loop I was able to build my game around it and it worked flawlessly.

Movement

For player movement I once again inferred to my previous experience working with the Godot game engine that uses the language GDscript. All I had to do is use the same procedure for keyboard and mouse inputs but port it to the java language for this project which was trivial.

Camera

Making the camera that follows the player around was an interesting challenge. During the coding I had found a neat little trick that would prevent the camera from going out of bounds.

```
public void checkBlankSpace(){
    if(offSet.getX() < 0)
        offSet.setX(0);
    if(offSet.getY() < 0)
        offSet.setY(0);
    if(offSet.getX() > World.WIDTH*Tile.TILESIZE - Window.WIDTH)
```

```
offSet.setX(World.WIDTH*Tile.TILESIZE - Window.WIDTH);

if(offSet.getY() > World.HEIGHT*Tile.TILESIZE - Window.HEIGHT)
offSet.setY(World.HEIGHT*Tile.TILESIZE - Window.HEIGHT);}
```

In this bit of code when ever the camera would go out of bounds I'd simply revert the offset to 0 or maximum.

Terrain

In the game, I've used 2 types of tiles, grass and dirt. The tiles are then arranged using a 2D matrix and the trees which are game objects are added after by using an add method. This is a relatively simple way of building the world with the limitation being there is nothing that prevents the player for leaving the play area.

Chapter-3: Graphics

Sprite based animation

The player and enemy animations are done using sprites which is a very common and very old way of animating characters. The idea being every frame of the animation are to be stored as a separate image and be assigned an index. The game simply loops through the indexes and displays the image corresponding to the index which the player perceives as animated characters. It should be noted that it's a cheap trick and uses a lot of memory as a result. All the sprite images are loaded into a separate texture package.

Buffer frames

Since the handler updates the game 60 times a second sometimes the key inputs may be swapped in the middle of a frame which wouldn't be displayed properly and cause visual lag. To mitigate this issue I've included a buffer frame system that preloads 3 frames in advance so that the transition from player inputs are smoother which leads to a smoother game experience. Adding more than 3 frames in buffer has no real benefits and can cause input lag.

Chapter-4: Conclusion

Limitations

This is NOT a complete game as there is much left to do. But the project as it stands can provide a base line for others to build upon. There are bugs and limitations in the game, the most notable being that Health is only visual and does not kill the player as of yet. And that the player can move out of bounds with no way to stop him.

Future work

Given that my computer was broken halfway through the project and I had to port it over to a borrowed PC there is a lot more I had hoped to achieve but couldn't. For future works I'd like to implement a proper death condition and add a level up mechanic with roguelike upgrades. But the fact that I have accomplished this project with limited resources and various setbacks is something I can be proud of.

Thank you.