โครงงาน

ZombieRider 2D

จัดทำโดย

6704062611328 ณพัฒน์ ลัทธิ

เสนอ

ผู้ช่วยศาสตราจารย์ ดร. สถิตย์ ประสมพันธ์

วิชา 040613204 Object-Oriented Programming

ภาคเรียนที่ 1/2567

ภาควิชาวิทยาการคอมพิวเตอร์และสารสนเทศ คณะวิทยาศาสตร์ประยุกต์

มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ

บทที่ 1

บทนำ

1.1 ที่มาและความสำคัญ

โครงงานนี้จัดขึ้นเพื่อวัดผลความสามารถในการเรียนวิชา Object Oriented Programming โดยการนำเรื่องที่เรียนมาสร้างเป็นชิ้นงานในรูปแบบของเกมโดยใช้แนวคิดการเขียนโปรกแกรมแบบเชิงวัตถุและยังช่วยให้ผู้จัดทำเรียนรู้อุปกรณ์และเครื่องมือ

1.2 วัตถุประสงค์

1.2.1 เพื่อนำเนื้อหาที่เรียนมาประยุกต์ใช้ทำงานจริง

1.2.2 เพื่อสะสมประสบการณ์การเขียนโปรแกรมแบบเชิงวัตถุ

1.3 ขอบเขตของโครงงาน

1.3.1 ใช้ภาษา JAVA ในการสร้างเกม

1.3.2 ใช้หลักการการเขียนโปรแกรมแบบเชิงวัตถุ

1.4 ประโยชน์ที่ได้รับจากโครงงาน

1.4.1 ได้เรียนรู้ประสบการณ์การเขียนโปรแกรมแบบเชิงวัตถุ

1.4.2 ได้นำเนื้อหาที่เรียนมาประยุกต์ใช้ในการทำงาน

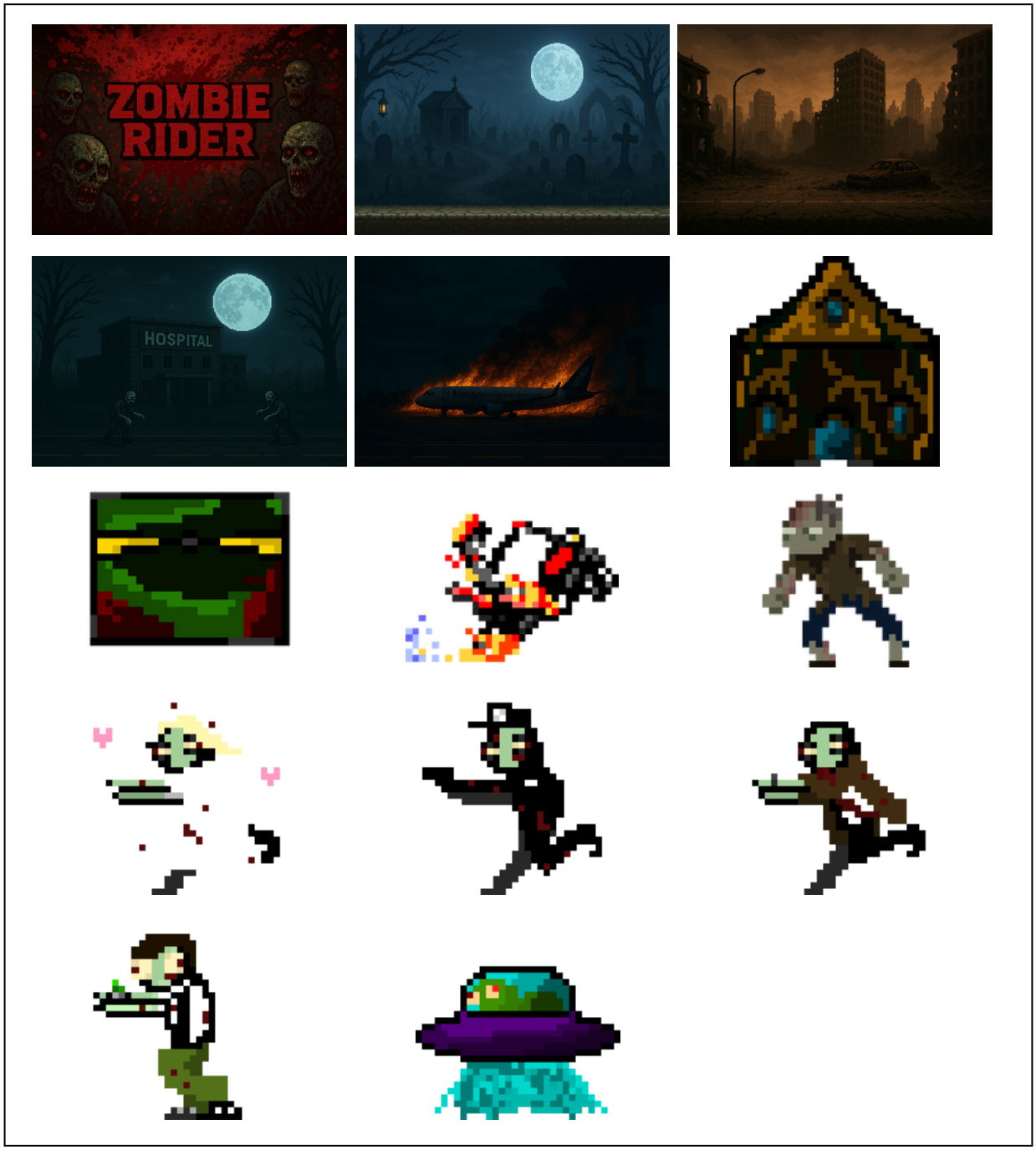
1.4.3 ได้เรียนรู้การทำงานแบบเป็นระบบมีแบบแผน

1.4.4 ได้เรียนรู้เครื่องมือต่างๆ

1.5 แผนการทำงาน

ลาดับ	รายการ	22-15	16-20	21-31
1	หารูปตัวละครและทำกราฟิกต่างๆ			
2	ศึกษาเอกสารและข้อมูลที่เกี่ยวข้อง			
3	ลงมือเขียนโปรแกรม			
4	จัดทำเอกสาร			
5	ตรวจสอบและแก้ไขข้อผิดพลาด			

Storyboard



บทที่ 2

การพัฒนา

2.1 รายละเอียดเกม

เป็นเกมต่อสู้เอาชีวิตรอดจากการแพร่ระบาดของเชื้อซอมบี้โดยได้รับบทบาทเป็นนักขี่มอเตอร์ไซค์ที่เอาชีวิตรอดจากซอมบี้กลายพันธ์

2.2 วิธีการเล่น

ผู้เล่นต้องเอาชีวิตรอดด้วยการยิงซอมบี้ โดยการใช้เม้าส์ และ คลิกซ้ายเพื่อเป็นการยิงซอมบี้โดยผู้เล่นจะมีพลังชีวิต 3 ชีวิตที่ต้องระวังอย่าให้พลังชีวิตหมดและคอยเก็บกระสุนเผื่อมาใช้กำจัดซอมบี้ และ ผู้เล่นสามารถยกล้อเผื่อกำจัดซอมบี้ได้ โดยการกด ลูกศรขวา แต่จะไม่สามารถกำจัดซอมบี้กลายพันธ์บางตัวด้วย

2.3 Class Diagram

มีทั้งหมด 14 คลาส

1.Main - Set Frame Size

2.GamePanel - Overall Function

3.StartMenu - Menu

4.Blackground - Blackground

5.Player - Rider data

6.Zombie - Zombie data

7.Shop - Shop data

8.Boss - Boss data

9.FlyingZombie - FlyingZombie data

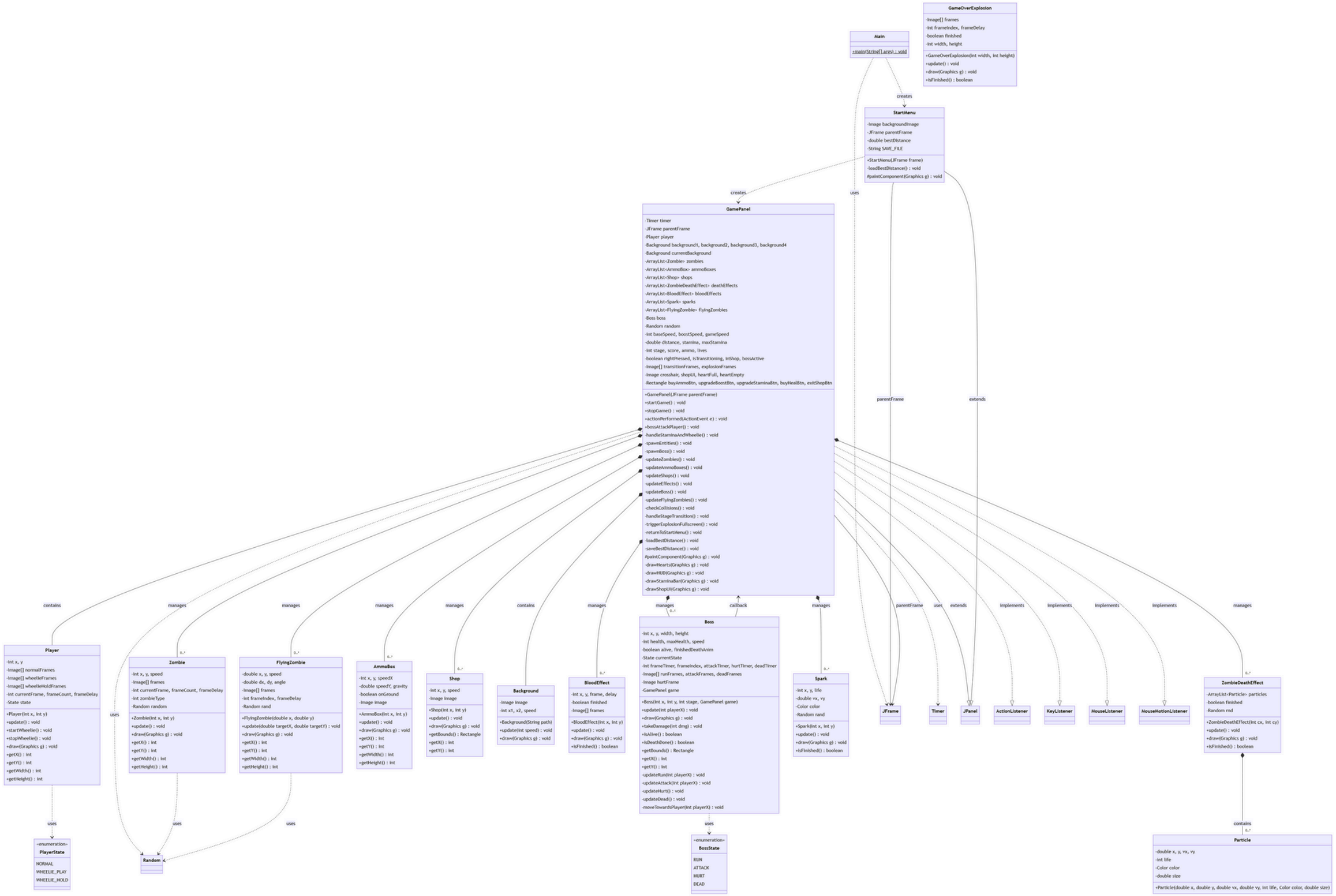
10.AmmoBox - AmmoBox data

11.BloodEffect - BloodEffect data

12.GameOverExplosion - GameOverExplosion data

13.Spark - Spark data

14.ZombieDeathEffect - ZombieDeathEffect data


Main
+main(String[] args) : void
GameOverExplosion
-Image[] frames
-int frameIndex, frameDelay
-boolean finished
-int width, height
+GameOverExplosion(int width, int height)
+update() : void
+draw(Graphics g) : void
+isFinished() : boolean
creates
StartMenu
-Image backgroundImage
-JFrame parentFrame
-double bestDistance
-String SAVE_FILE
+StartMenu(JFrame frame)
-loadBestDistance() : void
#paintComponent(Graphics g) : void
creates
uses
parentFrame
extends
GamePanel
-Timer timer
-JFrame parentFrame
-Player player
-Background background1, background2, background3, background4
-Background currentBackground
-ArrayList<Zombie> zombies
-ArrayList<AmmoBox> ammoBoxes
-ArrayList<Shop> shops
-ArrayList<ZombieDeathEffect> deathEffects
-ArrayList<BloodEffect> bloodEffects
-ArrayList<Spark> sparks
-ArrayList<FlyingZombie> flyingZombies
-Boss boss
-Random random
-int baseSpeed, boostSpeed, gameSpeed
-double distance, stamina, maxStamina
-int stage, score, ammo, lives
-boolean rightPressed, isTransitioning, inShop, bossActive
-Image[] transitionFrames, explosionFrames
-Image crosshair, shopUI, heartFull, heartEmpty
-Rectangle buyAmmoBtn, upgradeBoostBtn, upgradeStaminaBtn, buyHealBtn, exitShopBtn
+GamePanel(JFrame parentFrame)
+startGame() : void
+stopGame() : void
+actionPerformed(ActionEvent e) : void
+bossAttackPlayer() : void
-handleStaminaAndWheelie() : void
-spawnEntities() : void
-spawnBoss() : void
-updateZombies() : void
-updateAmmoBoxes() : void
-updateShops() : void
-updateEffects() : void
-updateBoss() : void
-updateFlyingZombies() : void
-checkCollisions() : void
-handleStageTransition() : void
-triggerExplosionFullscreen() : void
-returnToStartMenu() : void
-loadBestDistance() : void
-saveBestDistance() : void
#paintComponent(Graphics g) : void
-drawHearts(Graphics g) : void
-drawHUD(Graphics g) : void
-drawStaminaBar(Graphics g) : void
-drawShopUI(Graphics g) : void
contains
manages
manages
manages
manages
contains
manages
manages
0..1
callback
manages
parentFrame
uses
extends
Implements
Implements
Implements
Implements
manages
Player
-int x, y
-Image[] normalFrames
-Image[] wheelieFrames
-Image[] wheelieHoldFrames
-int currentFrame, frameCount, frameDelay
-State state
+Player(int x, int y)
+update() : void
+startWheelie() : void
+stopWheelie() : void
+draw(Graphics g) : vold
+getX() : int
+getY() : int
+getWidth() : int
+getHeight() : int
Zombie
-int x, y, speed
-Image[] frames
-int currentFrame, frameCount, frameDelay
-int zombieType
-Random random
+Zombie(int x, int y)
+update() : void
+draw(Graphics g) : void
+getX() : int
+getY() : int
+getWidth() : int
+getHeight() : int
FlyingZombie
-double x, y, speed
-double dx, dy, angle
-Image[] frames
-int frameIndex, frameDelay
-Random rand
+FlyingZombie(double x, double y)
+update(double targetX, double targetY) : void
+draw(Graphics g) : void
+getX() : int
+getY() : int
+getWidth() : int
+getHeight() : int
AmmoBox
-int x, y, speedX
-double speedY, gravity
-boolean onGround
-Image image
+AmmoBox(int x, int y)
+update() : void
+draw(Graphics g) : void
+getX() : int
+getY() : int
+getWidth() : int
+getHeight() : int
Shop
-int x, y, speed
-Image image
+Shop(int x, int y)
+update() : void
+draw(Graphics g) : void
+getBounds() : Rectangle
+getX() : int
+getY() : int
Background
-Image image
-int x1, x2, speed
+Background(String path)
+update(int speed) : void
+draw(Graphics g) : void
BloodEffect
-int x, y, frame, delay
-boolean finished
-Image[] frames
+BloodEffect(int x, int y)
+update() : void
+draw(Graphics g) : void
+isFinished() : boolean
Boss
-int x, y, width, height
-int health, maxHealth, speed
-boolean alive, finishedDeathAnim
-State currentState
-int frameTimer, frameIndex, attackTimer, hurtTimer, deadTimer
-Image[] runFrames, attackFrames, deadFrames
-Image hurtFrame
-GamePanel game
+Boss(int x, int y, int stage, GamePanel game)
+update(int playerX) : void
+draw(Graphics g) : void
+takeDamage(int dmg) : void
+isAlive() : boolean
+isDeathDone() : boolean
+getBounds() : Rectangle
+getX() : int
+getY() : int
-updateRun(int playerX) : void
-updateAttack(int playerX) : void
-updateHurt() : void
-updateDead() : void
-moveTowardsPlayer(int playerX) : void
Spark
-int x, y, life
-double vx, vy
-Color color
-Random rand
+Spark(int x, int y)
+update() : void
+draw(Graphics g) : void
+isFinished() : boolean
JFrame
Timer
JPanel
ActionListener
KeyListener
MouseListener
MouseMotionListener
ZombieDeathEffect
-ArrayList<Particle> particles
-boolean finished
-Random rnd
+ZombieDeathEffect(int cx, int cy)
+update() : void
+draw(Graphics g) : void
+isFinished() : boolean
uses
uses
uses
uses
«enumeration»
PlayerState
NORMAL
WHEELIE_PLAY
WHEELIE_HOLD
Random
uses
«enumeration»
BossState
RUN
ATTACK
HURT
DEAD
contains
0..*
Particle
-double x, y, vx, vy
-int life
-Color color
-double size
+Particle(double x, double y, double vx, double vy, int life, Color color, double size)

2.4 รูปแบบการพัฒนา

Waterfall Model

-Planning

วางแผนการทำงานคิดไอเดียแนวเกมที่จะทำระบบคร่าวๆว่าควรมีอะไร

-Analyze

ตัวละคร , ด่าน , ซอมบี้ , ความสามารถตัวละคร , รูปแบบการเล่น

-Design

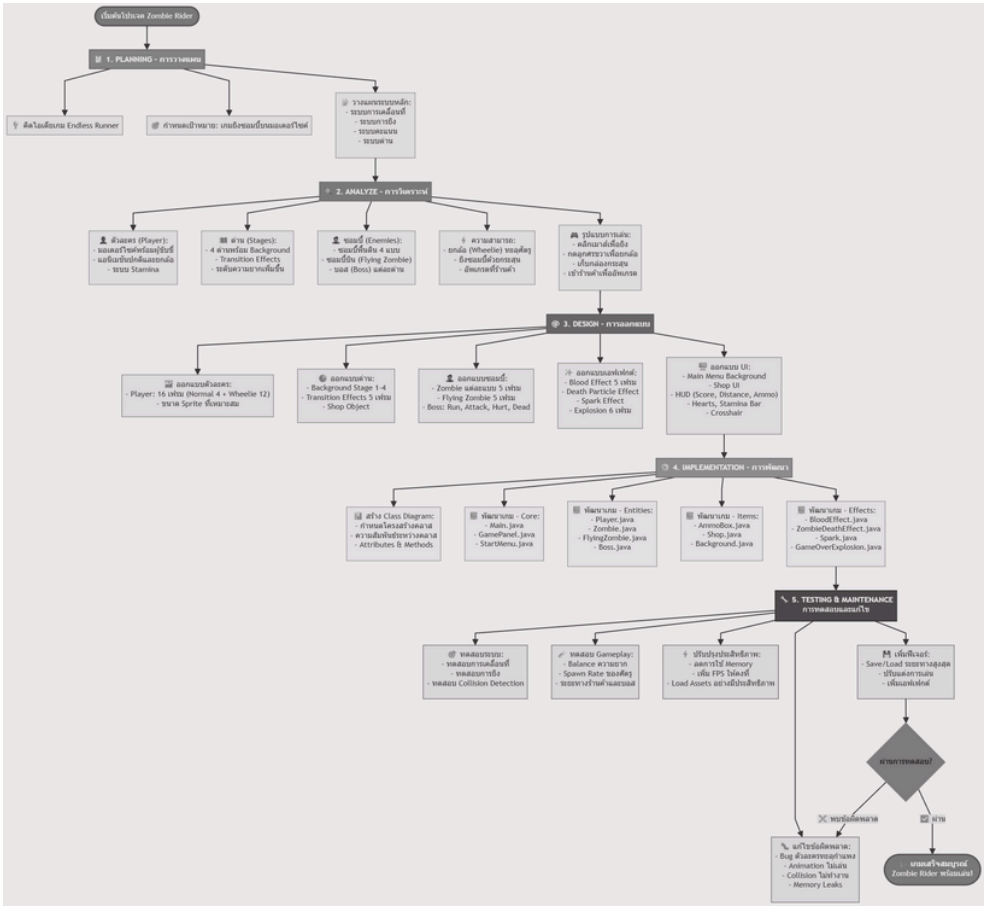
ออกแบบ ตัวละคร , ด่าน , ซอมบี้ , เอฟเฟกต์ , หน้าเมนู

-Implementation

คลาสไดอะแกรม , พัฒนาเกม

-Support && Resolve the errors

ทดสอบเกม , แก้ไขข้อผิดพลาด



Constructor

public class Main {
  public static void main(String[] args) {
  JFrame frame = new JFrame("Zombie Rider - Endless Runner");
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  frame.setSize(1536, 1024);
  frame.setResizable(false);
 	
  StartMenu menu = new StartMenu(frame);
  frame.add(menu);
 	
  frame.setVisible(true);
  }
}
	สร้างและเตรียมหน้าจอเมนูเริ่มต้นของเกม โดยเชื่อมต่อกับหน้าต่างหลัก (JFrame) และกำหนดองค์ประกอบต่าง ๆ เช่น ภาพพื้นหลังและปุ่มเริ่มเกม/ออกเกม

public GamePanel(JFrame parentFrame) {
  this.parentFrame = parentFrame;
  setFocusable(true);
  setPreferredSize(new Dimension(1536, 1024));
  setBackground(Color.BLACK);
 	
  addKeyListener(this);
  addMouseListener(this);
  addMouseMotionListener(this);
 	
  initGameObjects();
  flyingZombies = new ArrayList<>();
  loadAssets();
  loadBestDistance();
 	
  timer = new Timer(TARGET_DELAY, this);
}
	เตรียมค่าพื้นฐานทั้งหมดของเกมทั้งการตั้งค่าหน้าจอ โหลดภาพ สร้างอ็อบเจ็กต์หลัก และเชื่อมต่อ event ต่าง ๆ

public StartMenu(JFrame frame) {
this.parentFrame = frame;
loadBestDistance();
 	
backgroundImage = new
ImageIcon("assets/Menu_bg.png").getImage();
 	
setLayout(null);
 	
JButton startButton = new JButton("START");
startButton.setFont(new Font("Arial", Font.BOLD, 36));
startButton.setBackground(new Color(50, 205, 50));
startButton.setForeground(Color.BLACK);
startButton.setFocusPainted(false);
startButton.setBounds(650, 650, 250, 80); // x, y, width, height
 	
JButton exitButton = new JButton("EXIT");
exitButton.setFont(new Font("Arial", Font.BOLD, 36));
exitButton.setBackground(new Color(220, 20, 60));
exitButton.setForeground(Color.BLACK);
exitButton.setFocusPainted(false);
exitButton.setBounds(650, 760, 250, 80);
 	
add(startButton);
add(exitButton);
 	
startButton.addActionListener(e -> {
GamePanel gamePanel = new GamePanel(parentFrame);
parentFrame.setContentPane(gamePanel);
parentFrame.revalidate();
parentFrame.repaint();
gamePanel.startGame();
gamePanel.requestFocusInWindow();
});
exitButton.addActionListener(e -> System.exit(0));
}
	มีหน้าที่สร้างเมนูเริ่มต้นของเกม โดยโหลดภาพพื้นหลัง สร้างปุ่ม “START” และ “EXIT” พร้อมตั้งค่าการทำงานของแต่ละปุ่ม เพื่อให้ผู้เล่นสามารถเริ่มเกมหรือออกจากเกมได้

public Background(String path) {
image = new ImageIcon(path).getImage();
x1 = 0;
x2 = image.getWidth(null);
}
	ใช้สำหรับโหลดภาพพื้นหลังจากไฟล์ และตั้งค่าตำแหน่งเริ่มต้นของภาพ 2 ภาพ เพื่อให้สามารถทำเอฟเฟกต์พื้นหลังเลื่อนต่อเนื่อง (scrolling background) ได้ในเกม

public Player(int x, int y) {
this.x = x;
this.y = y;
 	
normalFrames = new Image[4];
for (int i = 0; i < 4; i++) {
normalFrames[i] = new ImageIcon("assets/Player_" + (i + 1) + ".png").getImage();
}
 	
wheelieFrames = new Image[8];
for (int i = 0; i < 8; i++) {
wheelieFrames[i] = new ImageIcon("assets/Player_" + (i + 5) + ".png").getImage();
}
 	
wheelieHoldFrames = new Image[4];
for (int i = 0; i < 4; i++) {
wheelieHoldFrames[i] = new ImageIcon("assets/Player_" + (i + 13) + ".png").getImage();
}
}
	ใช้สำหรับ “กำหนดตำแหน่งเริ่มต้นและโหลดภาพอนิเมชันทั้งหมดของตัวผู้เล่น” ได้แก่ท่าปกติ ท่ายกล้อ เพื่อให้ตัวละครสามารถแสดงการเคลื่อนไหวที่สมจริงในเกม

public Zombie(int x, int y) {
this.x = x;
this.y = y;
 	
zombieType = 1 + random.nextInt(TOTAL_TYPES);
 	
int frameTotal = 5;
frames = new Image[frameTotal];
 	
for (int i = 0; i < frameTotal; i++) {
String path = "assets/zombie" + zombieType + "_" + (i + 1) + ".png";
frames[i] = new ImageIcon(path).getImage();
}
 	
if (frames[0] != null) {
this.y -= frames[0].getHeight(null);
}
}
	ใช้สำหรับ สร้างและกำหนดค่าซอมบี้แต่ละตัวในเกม โดยสุ่มประเภทของซอมบี้ (zombieType), โหลดภาพอนิเมชัน 5 เฟรมตามประเภทนั้น และปรับตำแหน่งให้ตรงกับพื้น เพื่อเตรียมให้ซอมบี้พร้อมถูกวาดและเคลื่อนไหวในเกม

public Shop(int x, int y) {
this.x = x;
this.y = y;
image = new ImageIcon("assets/shop_obj.png").getImage();
}
	ใช้สำหรับ สร้างวัตถุร้านค้า (Shop) ในเกม โดยกำหนดตำแหน่ง (x, y) และโหลดภาพร้านจากไฟล์ shop_obj.png เพื่อแสดงในฉากเมื่อผู้เล่นถึงจุดร้านค้า

public Boss(int x, int y, int stage, GamePanel game) {
this.x = x;
this.y = y;
this.width = 256;
this.height = 256;
this.maxHealth = 100 + (stage * 50);
this.health = maxHealth;
this.game = game;
 	
runFrames = new Image[]{
new ImageIcon("assets/ZombieBoss_run2.png").getImage(),
new ImageIcon("assets/ZombieBoss_run3.png").getImage(),
new ImageIcon("assets/ZombieBoss_run4.png").getImage(),
new ImageIcon("assets/ZombieBoss_run5.png").getImage(),
new ImageIcon("assets/ZombieBoss_run6.png").getImage(),
new ImageIcon("assets/ZombieBoss_run7.png").getImage(),
new ImageIcon("assets/ZombieBoss_run8.png").getImage(),
new ImageIcon("assets/ZombieBoss_run9.png").getImage()
};
 	
hurtFrame=newImageIcon("assets/ZombieBoss_hurt.png").getImage();
 	
deadFrames = new Image[]{
newImageIcon("assets/ZombieBoss_dead1.png").getImage(),
newImageIcon("assets/ZombieBoss_dead2.png").getImage(),
newImageIcon("assets/ZombieBoss_dead3.png").getImage(),
newImageIcon("assets/ZombieBoss_dead4.png").getImage(),
newImageIcon("assets/ZombieBoss_dead5.png").getImage(),
newImageIcon("assets/ZombieBoss_dead6.png").getImage()
};
}
	ใช้สำหรับ สร้างบอส (Boss) ของแต่ละด่าน โดยกำหนดตำแหน่ง ขนาด พลังชีวิตตามระดับด่าน (stage), เชื่อมโยงกับ GamePanel เพื่อเข้าถึงข้อมูลเกม และโหลดภาพอนิเมชันทั้งหมดของบอส (ท่าวิ่ง ท่าโดนโจมตี และท่าตาย) เพื่อใช้แสดงผลในเกม

public FlyingZombie(double x, double y) {
this.x = x;
this.y = y;
this.speed = 5 + rand.nextDouble() * 3;
 	
frames = new Image[5];
for (int i = 0; i < frames.length; i++) {
frames[i] = new ImageIcon("assets/zombie_fly" + (i + 1) + ".png").getImage();
}
}
	ใช้สำหรับ สร้างซอมบี้บิน (FlyingZombie) โดยกำหนดตำแหน่งเริ่มต้น ความเร็วแบบสุ่ม และโหลดภาพอนิเมชัน 5 เฟรม เพื่อให้สามารถเคลื่อนไหวและแสดงภาพบินในเกมได้อย่างสมจริง

public AmmoBox(int x, int y) {
this.x = x;
this.y = y;
image = new ImageIcon("assets/ammo_box.png").getImage();
if (image != null) this.y -= image.getHeight(null);
}
	ใช้สำหรับ สร้างกล่องกระสุน (AmmoBox) โดยกำหนดตำแหน่งและโหลดภาพกล่องกระสุนจากไฟล์ ammo_box.png พร้อมปรับตำแหน่งให้วางอยู่บนพื้นอย่างเหมาะสมเมื่อแสดงในฉากเกม

public BloodEffect(int x, int y) {
this.x = x;
this.y = y;
frames = new Image[5];
for (int i = 0; i < frames.length; i++) {
frames[i] = new ImageIcon("assets/blood" + (i + 1) + ".png").getImage();
}
}
	ใช้สำหรับ สร้างเอฟเฟกต์เลือด (BloodEffect) เมื่อซอมบี้หรือบอสถูกโจมตี โดยกำหนดตำแหน่งที่เกิดเอฟเฟกต์และโหลดภาพอนิเมชันเลือด 5 เฟรม เพื่อแสดงผลตอนโดนยิงหรือชนในเกม

public GameOverExplosion(int width, int height) {
this.width = width;
this.height = height;
frames = new Image[]{
new ImageIcon("assets/explosion_full1.png").getImage(),
new ImageIcon("assets/explosion_full2.png").getImage(),
new ImageIcon("assets/explosion_full3.png").getImage(),
new ImageIcon("assets/explosion_full4.png").getImage(),
new ImageIcon("assets/explosion_full5.png").getImage(),
new ImageIcon("assets/explosion_full6.png").getImage(),
};
}
	ใช้สำหรับ สร้างเอฟเฟกต์ระเบิดเต็มจอ (GameOverExplosion) เมื่อผู้เล่นแพ้หรือเกมจบ โดยกำหนดขนาดของภาพและโหลดภาพอนิเมชันระเบิด 6 เฟรม เพื่อใช้แสดงเอฟเฟกต์ตอนจบเกม

public Spark(int x, int y) {
this.x = x + rand.nextInt(10) - 5;
this.y = y + rand.nextInt(10) - 5;
vx = rand.nextDouble() * 2 - 1;
vy = -rand.nextDouble() * 2;
color = new Color(255, 200 + rand.nextInt(55), 0);
}
	ใช้สำหรับ สร้างเอฟเฟกต์ประกายไฟ (Spark) เมื่อยกล้อ

public ZombieDeathEffect(int cx, int cy) {
particles = new ArrayList<>();
for (int i = 0; i < PARTICLE_COUNT; i++) {
double angle = rnd.nextDouble() * Math.PI * 2;
double speed = 2.5 + rnd.nextDouble() * 3.5;
double vx = Math.cos(angle) * speed;
double vy = Math.sin(angle) * speed - (1.0 + rnd.nextDouble()*1.5);
int life = 28 + rnd.nextInt(12);
Color c = i % 3 == 0 ? new Color(255,160,0) : new Color(255,80,30);
double size = 6 + rnd.nextDouble()*8;
particles.add(new Particle(cx, cy, vx, vy, life, c, size));
}
}
	ใช้สำหรับ สร้างเอฟเฟกต์การตายของซอมบี้ (ZombieDeathEffect)กำหนดทิศทาง ความเร็ว สี และขนาด เพื่อให้เกิดเอฟเฟกต์ระเบิดเลือดหรือเศษเนื้อกระจายอย่างสมจริงเมื่อซอมบี้ถูกทำลาย

Encapsulation

ทุก Class เป็น Encapsulation เพราะ มีการประกาศตัวแปรต่าง ๆ เป็น private เพื่อจำกัดการเข้าถึงจากภายนอกคลาส ซึ่งเป็นหลักของ Encapsulation เพราะช่วยป้องกันไม่ให้ข้อมูลสำคัญถูกแก้ไขโดยตรง และบังคับให้เข้าถึงผ่าน method เท่านั้น

Inheritance

public class GamePanel extends JPanel implements ActionListener, KeyListener, MouseListener, MouseMotionListener {
...
@Override
protected void paintComponent(Graphics g) {
super.paintComponent(g);
...
}
...
}
	GamePanel สืบทอดจาก JPanel เพื่อใช้งานและปรับแต่งพฤติกรรมการวาดภาพ (paintComponent) ใหม่ให้เหมาะกับการแสดงผลของเกม โดยยังสามารถใช้ฟังก์ชันพื้นฐานของ JPanel

public class StartMenu extends JPanel {
...
}
	StartMenu สืบทอดจาก JPanel เพื่อใช้ฟังก์ชันและพฤติกรรมของแผงวาดภาพกราฟิก (Panel) ของ Swing ได้

Polymorphism

public class GamePanel extends JPanel implements ActionListener, KeyListener, MouseListener, MouseMotionListener {
...
@Override
protected void paintComponent(Graphics g) { ... }
@Override
public void actionPerformed(ActionEvent e) { ... }
@Override
public void keyPressed(KeyEvent e) { ... }
@Override
public void keyReleased(KeyEvent e) { ... }
@Override
public void mousePressed(MouseEvent e) { ... }
@Override
public void mouseMoved(MouseEvent e) { ... }
...
}
	GamePanel implements หลายอินเทอร์เฟซและเขียนเมธอดที่มีชื่อเดียวกัน (actionPerformed, keyPressed, mousePressed, ฯลฯ) ขึ้นมาใหม่ เพื่อกำหนดพฤติกรรมเฉพาะของเกม และยังสามารถถูกใช้งานผ่านการอ้างอิงแบบชนิดแม่ (เช่น ActionListener) ได้

public class StartMenu extends JPanel {
...
@Override
protected void paintComponent(Graphics g) {
...
}
}
	StartMenu ได้เขียนทับเมธอด paintComponent() ของ JPanel เพื่อกำหนดพฤติกรรมการวาดภาพในแบบเฉพาะของเมนู และเมื่อรันจริง ระบบจะเรียกใช้เมธอดของ StartMenu แทนของ JPanel ซึ่งเป็นตัวอย่างของ Dynamic Polymorphism

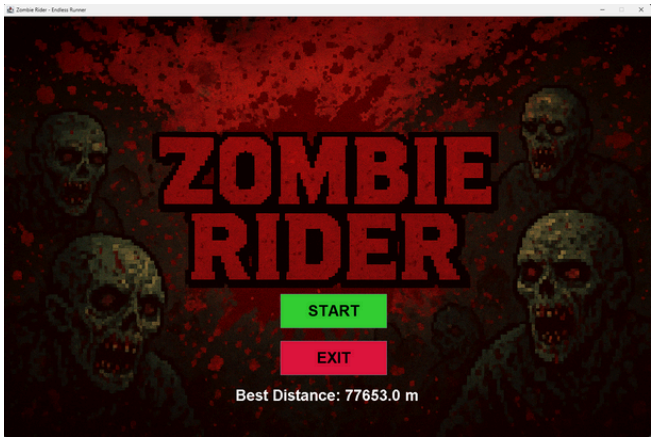
Composition

```
public class GamePanel extends JPanel implements
ActionListener, KeyListener, MouseListener, MouseMotionListener
{
  ...
        private Player player;
        private Boss boss = null;
        private ArrayList<Zombie> zombies;
        private ArrayList<AmmoBox> ammoBoxes;
        private ArrayList<Shop> shops;
        private ArrayList<ZombieDeathEffect> deathEffects;
        private ArrayList<BloodEffect> bloodEffects;
        private ArrayList<Spark> sparks;
        private ArrayList<FlyingZombie> flyingZombies;
  ...
}
```

GamePanel ประกอบไปด้วยอ็อบเจ็กต์จากหลายคลาส (เช่น Player, Zombie, Shop, Effect) ซึ่งร่วมกันทำให้เกิดการทำงานของเกม โดยที่วัตถุเหล่านี้มีความสัมพันธ์แบบ “has-a” กับ GamePanel และไม่สามารถทำงานได้อย่างอิสระนอกคลาสนี้

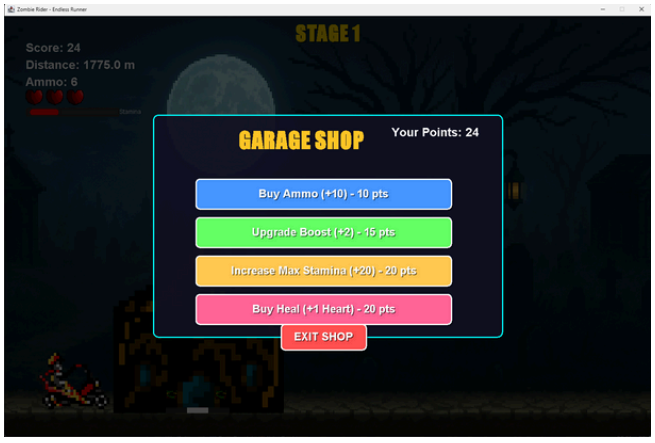
GUI

StartMenu



ประกอบไปด้วย Background ปุ่ม start , exit ข้อความ Best Distance จาก Class Background , StartMenu ที่มีการสร้างปุ่มจาก jButton

ShopMenu



ถูกสร้างมาจาก Method drawShopUi ใน Class GamePanel โดยการใช้ drawButton

Event Handling

@Override
public void actionPerformed(ActionEvent e) {
if (isReturningToMenu) {
return;
}
if (explosionActive) {
updateExplosion();
repaint();
return;
}
if (isGameOver) {
repaint();
return;
}
if (shopCooldown) {
shopCooldownTicks++;
if (shopCooldownTicks > 120) {
shopCooldown = false;
shopCooldownTicks = 0;
}
}
if (inShop) {
repaint();
return;
}
handleStaminaAndWheelie();
 	
distance += gameSpeed * 0.5;
player.update();
if (currentBackground != null)
currentBackground.update(gameSpeed);
 	
if (!bossActive && distance >= nextBossDistance) {
spawnBoss();
}
 	
handleStageTransition();spawnEntities();updateZombies();
updateAmmoBoxes();updateShops();updateEffects();
updateSparks();updateBloodEffects();updateBoss();
updateFlyingZombies();checkCollisions();
repaint();
}
	actionPerformed(ActionEvent e) ถูกเรียกอัตโนมัติทุกครั้งที่เกิด Event จาก Timer เพื่อจัดการอัปเดตสถานะเกมทั้งหมด เช่น การเคลื่อนไหวของผู้เล่น ซอมบี้ เอฟเฟกต์ และการตรวจการชน

```
@Override
public void mousePressed(MouseEvent e) {
    int mx = e.getX(), my = e.getY();
    for (int i = 0; i < flyingZombies.size(); i++) {
        FlyingZombie fz = flyingZombies.get(i);
        int fx = fz.getX() - fz.getWidth() / 2;
        int fy = fz.getY() - fz.getHeight() / 2;
        Rectangle fr = new Rectangle(fx, fy, fz.getWidth(), fz.getHeight());
        if (fr.contains(mx, my)) {
            bloodEffects.add(new BloodEffect(fz.getX(), fz.getY()));
            deathEffects.add(new ZombieDeathEffect(fz.getX(), fz.getY()));
            flyingZombies.remove(i--);
            score++;
            break;
        }
    }
    if (inShop) {
        if (buyAmmoBtn.contains(mx,my) && score >= 10) {
            ammo += 10; score -= 10;
        } else if (upgradeBoostBtn.contains(mx,my) && score >= 15) {
            boostSpeed += 2; score -= 15;
        } else if (upgradeStaminaBtn.contains(mx,my) && score >= 20) {
            maxStamina += 20; stamina = maxStamina; score -= 20; staminaCooldown = false;
        } else if (buyHealBtn.contains(mx,my) && score >= 20) {
            if (lives < 3) { lives++; score -= 20; }
        } else if (exitShopBtn.contains(mx,my)) {
            inShop = false;
            shopCooldown = true;
            setCursor(blankCursor);
            shopActive = false;
        }
        repaint();
        return;
    }
    if (isGameOver || explosionActive) return;
    if (ammo <= 0) return;
    ammo--;
    for (int i = 0; i < zombies.size(); i++) {
        Zombie z = zombies.get(i);
        Rectangle zr = new Rectangle(z.getX(), z.getY(), z.getWidth(), z.getHeight());
        if (zr.contains(mx,my)) {
            bloodEffects.add(new BloodEffect(z.getX(), z.getY()));
            deathEffects.add(new ZombieDeathEffect(z.getX()+z.getWidth()/2, z.getY()+z.getHeight()/2));
            if (random.nextInt(100) < 20) ammoBoxes.add(new AmmoBox(z.getX(), 900));
            zombies.remove(i--);
            score++;
            break;
        }
    }
    if (boss != null && boss.isAlive()) {
        Rectangle br = boss.getBounds();
        if (br.contains(mx,my)) {
            boss.takeDamage(25);
            bloodEffects.add(new BloodEffect(boss.getX(), boss.getY()));
        }
    }
    for (int i = 0; i < flyingZombies.size(); i++) {
        FlyingZombie fz = flyingZombies.get(i);
        Rectangle fr = new Rectangle(fz.getX(), fz.getY(), fz.getWidth(), fz.getHeight());
        if (fr.contains(mx, my)) {
            bloodEffects.add(new BloodEffect(fz.getX(), fz.getY()));
            deathEffects.add(new ZombieDeathEffect(fz.getX() + fz.getWidth() / 2, fz.getY() +
fz.getHeight() / 2));
            flyingZombies.remove(i--);
            score++;
            break;
        }
    }
    repaint();
}
```

เป็นเมธอดที่ใช้จัดการ “เหตุการณ์การคลิกเมาส์” (MouseEvent) โดยตรวจสอบการกระทำของผู้เล่นว่าคลิกโดนวัตถุใดในเกม แล้วตอบสนองด้วยการยิง ลบศัตรู แสดงเอฟเฟกต์ หรือทำงานในร้านค้า

@Override
public void mouseMoved(MouseEvent e) {
mouseX = e.getX();
mouseY = e.getY();
repaint();
}
	เป็นการตอบสนองต่อเหตุการณ์ “ผู้เล่นขยับเมาส์” โดยจับค่าตำแหน่งจาก MouseEvent e แล้วนำไปอัปเดตการแสดงผลในเกม เช่น การเลื่อน Crosshair ไปยิงซอมบี้

@Override
public void keyPressed(KeyEvent e) {
if (e.getKeyCode() == KeyEvent.VK_RIGHT && !inShop &&
stamina > 0 && !staminaCooldown) {
rightPressed = true;
player.startWheelie();
}
 	
if (isGameOver && e.getKeyCode() == KeyEvent.VK_R) {
resetGame();
}
}
	เป็นการตอบสนองต่อเหตุการณ์ “การกดปุ่มคีย์บอร์ด” โดยใช้ข้อมูลจาก KeyEvent เพื่อควบคุมการกระทำของผู้เล่นและสถานะของเกม เมื่อผู้เล่นกดลูกศรขวา จะยกล้อ

@Override
public void keyReleased(KeyEvent e) {
if (e.getKeyCode() == KeyEvent.VK_RIGHT) {
rightPressed = false;
player.stopWheelie();
}
}
	เป็นการตอบสนองต่อเหตุการณ์ “ผู้เล่นปล่อยปุ่ม” จากคีย์บอร์ดเมื่อผู้เล่นปล่อยลูกศรขวาจะเลิกยกล้อแล้วกลับมาท่าปกติ

Algorithm

```
@Override
public void actionPerformed(ActionEvent e) {
    if (isReturningToMenu) return;
    if (explosionActive) {
        updateExplosion();
        repaint();
        return;
    }
    if (isGameOver) {
        repaint();
        return;
    }
    if (shopCooldown) {
        shopCooldownTicks++;
        if (shopCooldownTicks > 120) {
            shopCooldown = false;
            shopCooldownTicks = 0;
        }
    }
    if (inShop) {
        repaint();
        return;
    }
    handleStaminaAndWheelie();
    distance += gameSpeed * 0.5;
    player.update();
    currentBackground.update(gameSpeed);

    if (!bossActive && distance >= nextBossDistance) {
        spawnBoss();
    }

    handleStageTransition();
    spawnEntities();

    updateZombies();
    updateAmmoBoxes();
    updateShops();
    updateEffects();
    updateSparks();
    updateBloodEffects();
    updateBoss();
    updateFlyingZombies();

    checkCollisions();
    repaint();
}
```

อัลกอริทึมนี้คือ “Game Loop หลัก” ที่ทำให้เกมเคลื่อนไหวได้ โดยทุกครั้งที่ Timer ส่ง Event มา

1 ตรวจสอบสถานะเกม

2 อัปเดตข้อมูล (ตัวละคร, ฉาก, บอส, เอฟเฟกต์)

3 ตรวจการชน

4 วาดภาพใหม่ (repaint())

```
private void checkCollisions() {
    Rectangle playerRect = new Rectangle(
        player.getX(), player.getY(),
        player.getWidth(), player.getHeight()
    );
    if (boss != null && boss.isAlive()) {
        Rectangle br = boss.getBounds();
        if (playerRect.intersects(br) && !isImmortal) {
            bossAttackPlayer();
        }
    }
    if (isImmortal) {
        for (Zombie z : zombies) {
            Rectangle zr = new Rectangle(z.getX(), z.getY(), z.getWidth(), z.getHeight());
            if (playerRect.intersects(zr)) {
                bloodEffects.add(new BloodEffect(z.getX(), z.getY()));
                deathEffects.add(new ZombieDeathEffect(...));
                zombies.remove(z);
                score++;
            }
        }
    } else {
        for (Zombie z : zombies) {
            Rectangle zr = new Rectangle(z.getX(), z.getY(), z.getWidth(), z.getHeight());
            if (playerRect.intersects(zr)) {
                zombies.remove(z);
                lives--;
                bloodEffects.add(new BloodEffect(z.getX(), z.getY()));
                if (lives <= 0) {
                    triggerExplosionFullscreen();
                }
                break;
            }
        }
    }
    for (AmmoBox b : ammoBoxes) {
        Rectangle br = new Rectangle(b.getX(), b.getY(), b.getWidth(), b.getHeight());
        if (playerRect.intersects(br)) {
            ammo += 1 + random.nextInt(10);
            ammoBoxes.remove(b);
        }
    }
    for (Shop s : shops) {
        if (!shopCooldown && playerRect.intersects(s.getBounds())) {
            inShop = true;
            rightPressed = false;
            isImmortal = false;
            setCursor(defaultCursor);
            break;
        }
    }
    for (FlyingZombie fz : flyingZombies) {
        Rectangle fr = new Rectangle((int)fz.getX(), (int)fz.getY(), fz.getWidth(),
fz.getHeight());
        if (playerRect.intersects(fr)) {
            flyingZombies.remove(fz);
            lives--;
            bloodEffects.add(new BloodEffect((int)fz.getX(), (int)fz.getY()));
            if (lives <= 0) triggerExplosionFullscreen();
            break;
        }
    }
}
```

อัลกอริทึม checkCollisions() ทำหน้าที่ ตรวจสอบการชนของผู้เล่นกับวัตถุทั้งหมดในเกมและอัปเดตผลลัพธ์ที่เกิดขึ้น เช่น เสียพลัง เก็บของ เพิ่มคะแนน หรือเข้าสู่ร้านค้า

private void spawnEntities() {
int zombieSpawnChance = ZOMBIE_BASE_SPAWN_CHANCE + (stage - 1) * 2;
// Stage 1: 3%, Stage 2: 5%, Stage 3: 7%, Stage 4: 9%
if (!shopActive && !isTransitioning && random.nextInt(100) < zombieSpawnChance) {
zombies.add(new Zombie(1600, 900));
}
if (!shopActive && !isTransitioning && random.nextInt(1000) < AMMOBOX_SPAWN_CHANCE) {
ammoBoxes.add(new AmmoBox(1600, 900));
}
if (!shopActive && !isTransitioning && distance >= nextShopDistance) {
shops.add(new Shop(1600, 530));
shopActive = true;
nextShopDistance += getShopDistanceGap();
}
if (!shopActive && !isTransitioning && random.nextInt(400) < 1 + stage) {
int spawnY = 300 + random.nextInt(300);
flyingZombies.add(new FlyingZombie(1600, spawnY));
}
}
	อัลกอริทึม spawnEntities() ทำหน้าที่ควบคุมการเกิดของวัตถุต่าง ๆ ในเกมตามเงื่อนไขและความน่าจะเป็น เช่น สร้างซอมบี้และไอเทมแบบสุ่มในแต่ละรอบของเกม

บทที่ 3

สรุปผลและข้อเสนอแนะ

3.1 ปัญหาที่พบระหว่างการพัฒนา

3.1.1 การออกแบบตัวละครภายในเกมเพราะต้องวาด frame by frame จึงค่อนข้างใช้เวลานาน

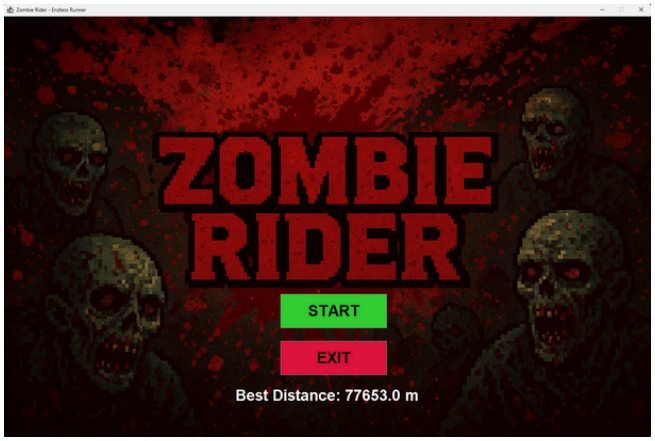
3.1.2 การออกแบบโครงสร้างเกม และ Implement ต่างๆ

3.1.3 การคำนวณสเกลของเกมให้สมมาตร

3.2 จุดเด่น

3.2.1 เป็นเกมที่ 2D ที่ออกแบบแบบ Pixel โดยจะออกแบบตัวละครภายในเกมด้วยตัวเอง

3.2.2 มีระบบเก็บคะแนนสูงสุดที่เราทำได้ Best Distance



3.3 ข้อเสนอแนะ

3.3.1 เวลาที่ทำแลปเสร็จอยากให้อาจาร์ยมาเฉลยพร้อมอธิบายถึง Logic วิธีการคิดต่างๆภายในโปรแกรม

3.3.2 อยากให้อาจาร์ยทำคลิปวิดีโอย้อนหลังไว้สำหรับการสอนเผื่อคนที่ยังไม่เข้าใจจะได้นำมาศึกษาเพิ่มเติม