โครงงาน

Pixel Forest Runner

จัดทำโดย

6704062612138 นายกรินทร์ สุขสอาด

เสนอ

ผู้ช่วยศาสตราจารย์ สถิต ประสมพันธ์

โครงงานนี้เป็นส่วนหนึ่งของวิชา Object Oriented Programming

ภาควิชาวิทยาการคอมพิวเตอร์และสาระสนเทศ คณะวิทยาศาสตร์ประยุกต์

มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ

ภาคเรียนที่ 1 / 2568

บทที่ 1

บทนำ

1.1 ที่มาและความสำคัญ

โครงงานนี้เกิดจากความสนใจในการเล่นเกมของผู้จัดทำ โดยเฉพาะเกม Dinosaur Game ที่มักเล่นเมื่อไม่มีอินเทอร์เน็ต ซึ่งได้แรงบันดาลใจให้สร้างเกมของตนเองในชื่อ Pixel Forest Runner ขึ้นมา เพื่อใช้เป็นสื่อในการประยุกต์ความรู้จากรายวิชา Object Oriented Programming (OOP) การสร้างเกมนี้ยังเป็นโอกาสให้ผู้จัดทำได้ทดลองเขียนโปรแกรมจริง และสามารถเล่นร่วมกับเพื่อน ๆ เพื่อสร้างความสนุกสนานและเสริมสร้างความสัมพันธ์ที่ดี

1.2 ประเภทของโครงการ

Game Java 2D

1.3 วัตถุประสงค์

1.3.1 เพื่อนำเนื้อหาที่เรียนมาประยุกต์ใช้งานจริงผ่านการสร้างเกม
1.3.2 เพื่อสะสมประสบการณ์และทักษะในการเขียนโปรแกรมแบบ OOP

1.4 ขอบเขตของโครงงาน

1.4.1 ระบบ Game Loop สร้าง Game Loop ที่ทำงานอย่างคงที่ (Fixed FPS) เพื่อควบคุมการอัปเดตตรรกะของเกมและการวาดภาพ
1.4.2 การจัดการสถานะ เกมต้องมีระบบ State Machine เพื่อจัดการฉากต่างๆ อย่างเป็นระบบ ได้แก่ MenuState, SelectionState(หน้าเลือกตัวละครและเลือกด่าน), PlayingState (ฉากเล่นเกมจริง), GameOverState (ฉากจบเกม)

1.4.3 ระบบการเลือก (Selection System) ผู้เล่นต้องสามารถเลือกตัวละคร ได้ (เช่น Neo, Kenji, Boss) ผู้เล่นต้องสามารถเลือกด่าน (พื้นหลัง) ได้ 2 แบบ ("The Forest of Dawn" และ "The Forest of Dusk")

1.4.4 การเล่นเกม (Gameplay ผู้เล่น (Player) สามารถวิ่งและกระโดด (jump()), มีอุปสรรค (ObstacleManager) สุ่มปรากฏขึ้น (เช่น Rock, Bush), มีการตรวจสอบการชน (Collision Detection) ระหว่างผู้เล่นและอุปสรรค

1.5 ประโยชน์ที่ได้รับจากโครงงาน

1.5.1 ผู้จัดทำได้รับประสบการณ์จริงในการเขียนโปรแกรมแบบ OOP
1.5.2 ได้ทดลองนำเนื้อหาที่เรียนมาประยุกต์ใช้ในงานจริง และพัฒนาทักษะการสร้างเกมให้สามารถเล่นได้สองคน

1.6 แผนการทำงาน

ลำดับ	รายการ	1 – 15ก.ย.	15ก.ย.-15ต.ค.	15 - 28ต.ค.
1	หารูปตัวละครในเกม			
2	ศึกษาเอกสารและข้อมูลที่เกี่ยวข้อง			
3	เขียนโปรแกรม			
4	จัดทำเอกสาร			
5	ตรวจสอบและแก้ไขข้อผิดพลาด			

บทที่ 2

การพัฒนา

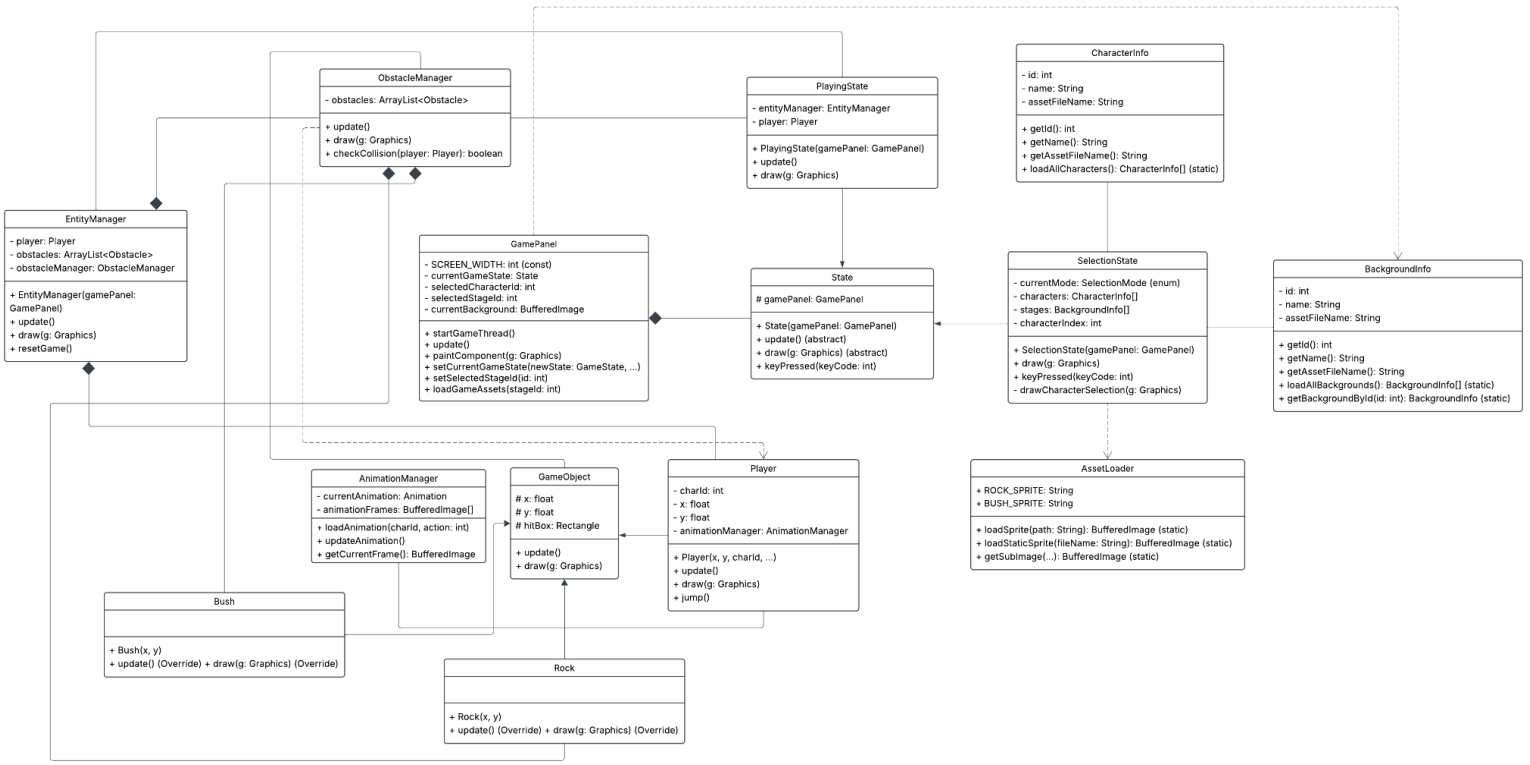
2.1 รายละเอียดเกม

เกม Pixel Forest Runner เป็นเกมวิ่งผจญภัยในป่าพิกเซลที่เต็มไปด้วยความสนุกและความท้าทาย ผู้เล่นจะควบคุมตัวละครหลักที่วิ่งอัตโนมัติจากซ้ายไปขวา ตลอดเส้นทางเต็มไปด้วยอุปสรรคสองประเภท คือ ก้อนหิน และ พุ่มไม้ ผู้เล่นต้องใช้ทักษะในการ กดกระโดด เพื่อหลบสิ่งกีดขวางแต่ละชนิด หากชนกับอุปสรรค ตัวละครจะเสียพลังชีวิต และเกมจะจบลงเมื่อพลังชีวิตหมด

2.2 วิธีการเล่น

เกมนี้มีระบบ พลังชีวิต (HP bar) เริ่มต้นที่ 100 หน่วย และคะแนน (Score) ที่เพิ่มขึ้นตามระยะทางที่ตัวละครวิ่งไป การเล่นเกมง่ายและสนุกเพียงใช้ ปุ่ม Space bar เพื่อกระโดดข้ามก้อนหินและพุ่มไม้ที่อยู่ข้างหน้า การวิ่งอัตโนมัติของตัวละครทำให้ผู้เล่นต้องมีสมาธิและความเร็วในการตอบสนองเพื่อหลีกเลี่ยงอุปสรรคและสะสมคะแนนให้ได้มากที่สุด

2.3 Class Diagram



2.4 รูปแบบการพัฒนา

ใช้การพัฒนาในรูปแบบ Waterfall model เพราะมีรูปแบบและแผนการดำเนินงานที่ตายตัว

2.4.1 Planning กำหนดแผนการดำเนินการ

2.4.1.1 ทำแผนการพัฒนา

2.4.2 Analyze วิเคราะห์รูปแบบเกม

2.4.2.1 ตัวละคร ด่าน
2.4.2.2 ความสามารถของตัวละคร
2.4.2.3 วิธีการเล่น

2.4.3 Design ออกแบบส่วนประกอบต่างๆ

2.4.3.1 ตัวละคร
2.4.3.2 ด่าน
2.4.3.3 หน้าเมนูต่างๆ

2.4.4 Implementation จัดทำเอกสารและพัฒนาโปรแกรม

2.4.4.1 ทำ Class Diagram
2.4.4.1 เริ่มพัฒนาเกม

2.4.5 support and security แก้ไขข้อผิดพลาดต่างๆ

2.5.4.1 ทดสอบเกม
2.5.4.1 แก้ไขข้อผิดพลาดต่างๆ

2.5 การทำงานของส่วนต่างๆ

2.5.1 Constructor

public GameObject(int x, int y, int width, int height) {
this.x = x;
this.y = y;
this.width = width;
this.height = height;
this.hitboxOffsetX = 0;
this.hitboxOffsetY = 0;
}
	ใช้สร้าง วัตถุทั่วไปในเกม เช่น ตัวละคร สิ่งกีดขวาง ฯลฯ มีตำแหน่ง (x, y) และขนาด (width, height) สร้าง hitbox ขึ้นมาทันที เพื่อใช้ตรวจจับการชน Player จะ override หรือปรับ hitbox เองภายหลัง

public Obstacle(int x, int y, int width, int height,
int hitboxOffsetX, int hitboxOffsetY,
int hitboxWidth, int hitboxHeight) {
super(x, y, width, height, hitboxOffsetX, hitboxOffsetY, hitboxWidth, hitboxHeight);
}
	ใช้สร้าง สิ่งกีดขวาง (Obstacle) ที่มี hitbox เฉพาะของมัน เรียก super() เพื่อให้คลาสแม่ (GameObject) จัดการสร้าง hitbox ให้ตามค่าที่ส่งมา ทำให้สิ่งกีดขวางแต่ละประเภท (ก้อนหิน, พุ่มไม้) สามารถมีขนาด hitbox แตกต่างกันได้

public Player(int x, int y, int width, int height, GamePanel gamePanel) {
super(x, y, width, height);
this.gamePanel = gamePanel;
this.currentState = PlayerState.RUNNING;
int hitboxOffsetX = 44;
int hitboxWidth = 40;
int hitboxOffsetY = 15;
int hitboxHeight = 105;
this.hitbox = new Rectangle(x + hitboxOffsetX, y + hitboxOffsetY, hitboxWidth, hitboxHeight);
this.selectedInfo = loadAnimations(gamePanel.getSelectedCharacterId());
	เรียก super() เพื่อกำหนดตำแหน่งและขนาดจาก GameObject เก็บ reference ของ gamePanel (ใช้เข้าถึงข้อมูลในเกม เช่น ด่าน, ตัวละครที่เลือก) ตั้งสถานะเริ่มต้นเป็น “วิ่งอยู่” (RUNNING) ปรับ hitbox เอง โหลดอนิเมชันของตัวละคร เรียกใช้ loadAnimations() เพื่อโหลดภาพเคลื่อนไหวของตัวละครที่เลือกไว้

this.animationManager = new AnimationManager(animRun); }	จากนั้นสร้าง animationManager มาควบคุมการแสดงผลการวิ่ง

2.5.2 Encapsulation

public class GamePanel extends JPanel implements Runnable {
public static final int SCREEN_WIDTH = 800;
public static final int SCREEN_HEIGHT = 600;
private static final int FPS = 60;
private Thread gameThread;
private Map<GameState, State> gameStates;
private State currentGameState;
private int selectedCharacterId = 1;
private int selectedStageId = BackgroundInfo.STAGE_FOREST_OF_DUSK;
private BufferedImage currentBackground;
…
}
	ในคลาสนี้ มีการกำหนดตัวแปรหลายตัวให้เป็น private เช่น gameThread, currentGameState, SCREEN_WIDTH และอื่นๆ ตัวแปรเหล่านี้จะถูก “ซ่อน” (Hide) ไว้ภายในคลาส `GamePanel` เท่านั้น ไม่สามารถเข้าถึงหรือแก้ไขได้จากคลาสอื่นโดยตรง

public class CharacterInfo {
public static final int CH1_ID = 1;
public static final int CH2_ID = 2;
public static final int CH3_ID = 3;
private final int id;
private final String name;
private final String runAsset;
private final String jumpAsset;
…
}
	CharacterInfo มีการใช้หลักการ Encapsulation อย่างชัดเจน โดยมีการกำหนดฟิลด์ภายในเป็น private ทั้งหมด เพื่อป้องกันไม่ให้คลาสภายนอกเข้าถึงหรือแก้ไขข้อมูลของตัวละครโดยตรง ข้อมูลสำคัญ เช่น id, name, runAsset และ jumpAsset

2.5.3 Composition

import entities.Obstacle;
import entities.Player;
public class EntityManager {
  private Player player;
  private List<Obstacle> obstacles;
  …
}
	คลาส EntityManager เป็นเจ้าของวัตถุ Player Player จะถูกสร้างขึ้นและถูกจัดการโดย EntityManager เท่านั้น ถ้า EntityManager หายไป (ถูกทำลายหรือปิดเกม) → Player ก็จะหายไปด้วย EntityManager เป็นเจ้าของ “รายการสิ่งกีดขวางทั้งหมดในเกม” การเพิ่ม / ลบ / อัปเดตสิ่งกีดขวาง จะทำผ่าน EntityManager เท่านั้น

import entities.Bush;
import entities.Rock;
public class ObstacleManager {
  private EntityManager entityManager;
  private long spawnTimer;
  private long spawnInterval = 2000;
  …
}
	มีการประกาศตัวแปร private EntityManager เพื่อใช้เป็นองค์ประกอบภายในสำหรับจัดการเอนทิตีต่าง ๆ ของเกม โดยเฉพาะสิ่งกีดขวาง (Obstacle) ซึ่งอาจมีทั้ง Bush และ Rock ภายในเกม เมื่อคลาส ObstacleManager ถูกสร้างขึ้น จะสร้างและควบคุมอ็อบเจ็กต์ของ EntityManager ไปพร้อมกัน

2.5.4 Polymorphism

public class MenuState extends State {
…
@Override
public void draw(Graphics g) {
g.setColor(Color.WHITE);
g.setFont(new Font("Arial", Font.BOLD, 60));
g.drawString("Pixel Forest Runner", 100, 200);
g.setFont(new Font("Arial", Font.PLAIN, 30));
for (int i = 0; i < options.length; i++) {
if (i == currentSelection) {
g.setColor(Color.YELLOW);
} else {
g.setColor(Color.WHITE);
}
g.drawString(options[i], 300, 350 + (i * 50));
}
}
…
}
	ในคลาส MenuState มีการนำหลัก Polymorphism มาใช้โดยการ Override เมธอด draw(Graphics g) จากคลาสแม่ State เพื่อให้สามารถแสดงผลในรูปแบบเฉพาะของหน้าจอเมนู เช่น การวาดชื่อเกมและตัวเลือกเมนู เมื่อมีการเรียกใช้เมธอด draw() ผ่านตัวแปรประเภท State ระบบจะเลือกเรียกใช้เมธอดของคลาสลูกที่ตรงกับอ็อบเจ็กต์จริงในขณะนั้น ซึ่งเป็นการแสดงออกของพหุรูปตามแนวคิดของการเขียนโปรแกรมเชิงวัตถุ (OOP) ที่ช่วยให้โค้ดมีความยืดหยุ่นและขยายได้ง่ายโดยไม่ต้องแก้ไขโค้ดเดิมของคลาสอื่น

public class PlayingState extends State implements Runnable {
…
@Override
public void update() {
if (!entityManager.getPlayer().isAlive()) {
if (animationThread != null && animationThread.isAlive()) {
 	
entityManager.getPlayer().getAnimationManager().stop();
}
 	
gamePanel.setCurrentGameState(GameState.GAME_OVER, score);
	ในคลาส PlayingState มีการนำหลัก Polymorphism มาใช้โดยการ Override เมธอด update() จากคลาสแม่ State เพื่อให้พฤติกรรมของเกมในระหว่างเล่นแตกต่างจากสถานะอื่น ๆ โดยในเมธอดนี้จะตรวจสอบการตายของผู้เล่น การหยุดชั่วคราว และการอัปเดตตำแหน่งของอ็อบเจ็กต์ในเกม ซึ่งแสดงให้เห็นถึงการใช้พหุรูปเพื่อแยกพฤติกรรมของแต่ละสถานะออกจากกัน ทำให้โค้ดมีค

return;
}
if (isPaused) {
return;
}
entityManager.update();
obstacleManager.update();
}
…
}
	วามยืดหยุ่น สามารถเพิ่มสถานะใหม่ได้โดยไม่ต้องแก้ไขโค้ดเดิมของระบบหลัก เช่น GamePanel

2.5.5 Abstract

package gamestates;
import java.awt.Graphics;
import core.GamePanel;
public abstract class State {
protected GamePanel gamePanel;
public State(GamePanel gamePanel) {
this.gamePanel = gamePanel;
}
public abstract void update();
public abstract void draw(Graphics g);
public void keyPressed(int keyCode) {}
public void keyReleased(int keyCode) {}
}
	คลาส State ถูกกำหนดเป็น abstract class เพื่อใช้เป็นแม่แบบของทุกสถานะในเกม เช่น เมนู, เล่นเกม, และจบเกม โดยมี abstract method update() และ draw(Graphics g) เพื่อบังคับให้คลาสลูกต้องกำหนดพฤติกรรมของตนเอง ทำให้มั่นใจว่าทุก State มีโครงสร้างพื้นฐานเดียวกัน ในขณะเดียวกันยังสามารถใส่เมธอดทั่วไป เช่น keyPressed() และ keyReleased() ให้คลาสลูกเลือกใช้งานหรือ override ตามความเหมาะสม การออกแบบนี้ช่วยให้เกิด Polymorphism ทำให้โค้ดเกมยืดหยุ่นและขยายได้ง่าย

2.5.6 Inheritance

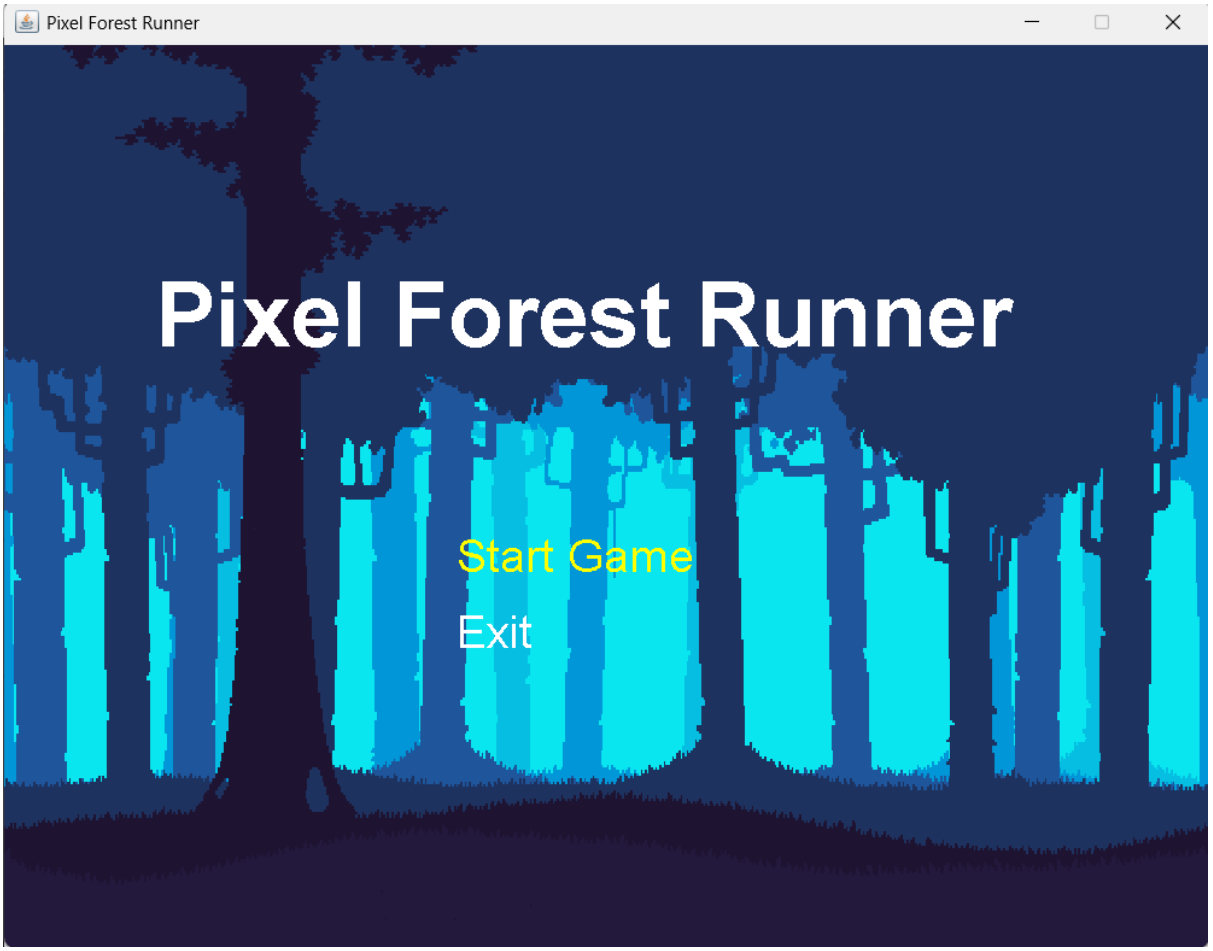
public abstract class GameObject {...} public abstract class Obstacle extends GameObject {...} public class Player extends GameObject {...} public class Bush extends Obstacle {...} public class Rock extends Obstacle {...}	ในเกม Pixel Forest Runner มีการใช้หลัก Inheritance (การสืบทอด) เพื่อสร้างความสัมพันธ์ระหว่างคลาสต่าง ๆ โดย GameObject เป็น abstract class แม่ที่กำหนดคุณสมบัติและพฤติกรรมพื้นฐานของวัตถุทั้งหมดในเกม เช่น ตำแหน่ง (x, y) และเมธอด update() จากนั้น Obstacle เป็น abstract class ลูกของ GameObject สำหรับสิ่งกีดขวางทั่วไป เช่น ก้อนหิน (Rock) และพุ่มไม้ (Bush) ซึ่งสืบทอดคุณสมบัติและเมธอดจากทั้ง Obstacle และ GameObject ส่วน Player ก็สืบทอดจาก GameObject เช่นกัน การออกแบบนี้ช่วยให้โค้ด นำกลับมาใช้ใหม่ได้ง่าย, ขยายระบบวัตถุได้สะดวก และแสดงความสัมพันธ์แบบ “is-a” อย่างชัดเจน

2.5.7 Thread

public class GamePanel extends JPanel implements Runnable {
…
public void startGameThread() {
gameThread = new Thread(this);
gameThread.start();
}
@Override
public void run() {
double drawInterval = 1_000_000_000.0 / FPS;
double delta = 0;
long lastTime = System.nanoTime();
long currentTime;
while (gameThread != null) {
currentTime = System.nanoTime();
delta += (currentTime - lastTime) / drawInterval;
lastTime = currentTime;
if (delta >= 1) {
update();
repaint();
delta--;
}
}
}
…
}
	ในโค้ดนี้มีการใช้ Thread เพื่อควบคุมการทำงานของเกมให้ดำเนินไปอย่างต่อเนื่องและเป็นอิสระจากส่วนของกราฟิกหลัก (JPanel) โดยคลาส GamePanel ทำการ implements Runnable เพื่อให้สามารถกำหนดการทำงานภายในเมธอด run() ได้ เมื่อเรียกใช้ startGameThread() โปรแกรมจะสร้าง Thread ใหม่และเริ่มทำงานผ่าน gameThread.start() ซึ่งจะไปเรียกเมธอด run() โดยอัตโนมัติ ภายใน run() มีการใช้ลูปหลักของเกม (Game Loop) ที่จะคอยอัปเดตสถานะของเกมด้วย update() และวาดภาพใหม่ด้วย repaint() ตามอัตราเฟรมที่กำหนดไว้ที่ 60 FPS ทำให้เกมมีการแสดงผลที่ลื่นไหลและไม่ค้าง

2.6 GUI

2.6.1 หน้า Menu

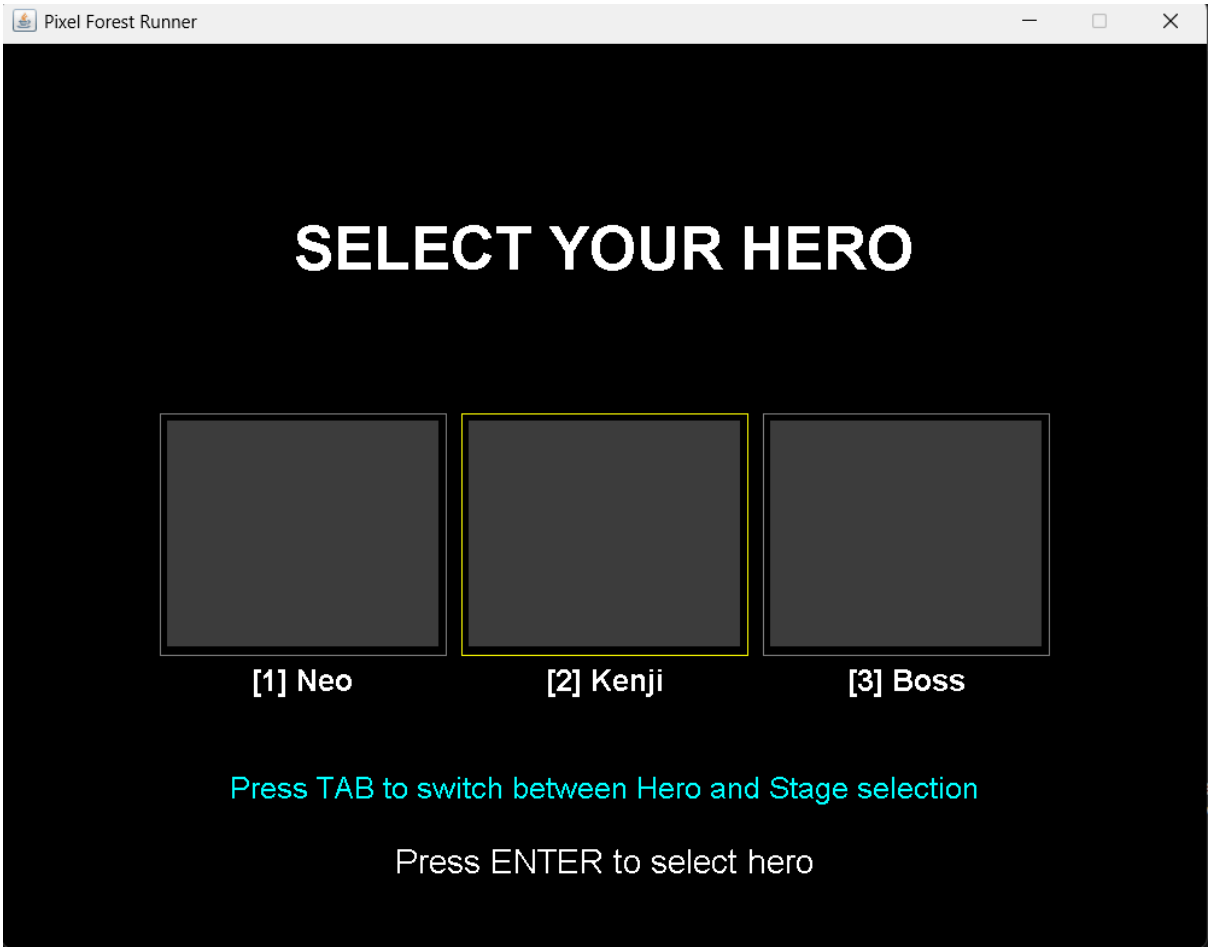


ประกอบไปด้วย

- Background Component (1 Component) - The Forest of Dusk
- Button Components (2 Components) - Start Game (สีเหลือง) ใช้เริ่มเกม และ Exit (สีขาว) → ใช้ปิดโปรแกรม

2.6.2 หน้า Selection Hero and Stage

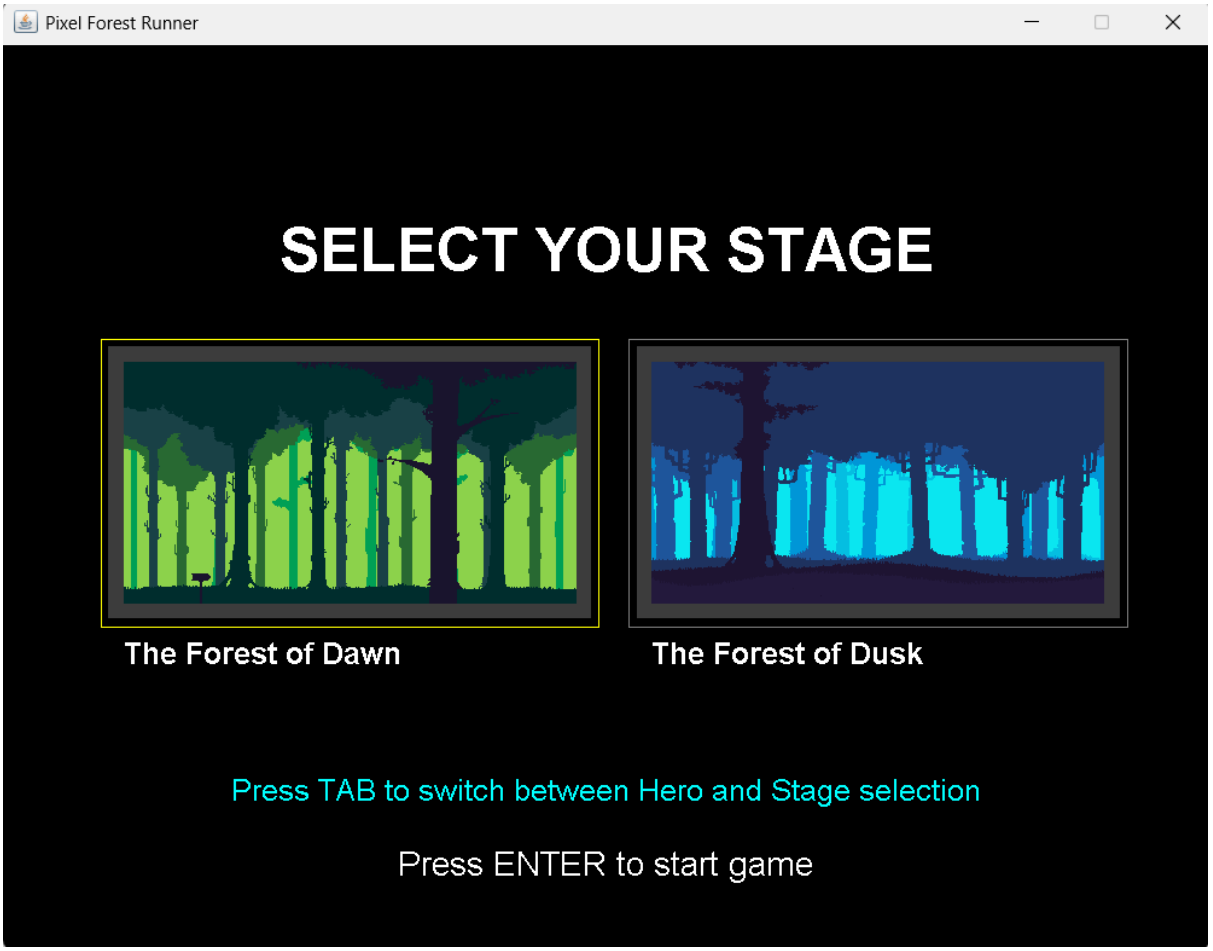
2.6.2.1 Selection Hero



ประกอบไปด้วย

- Background Component (1 Component) - Black Background
- Person Component (3 Component) - Neo, Kenji (มีกรอบสีเหลืองแสดงว่ากำลังถูกเลือกอยู่) และ Boss
- Text Component (2 Component) - ข้อความ “Press TAB to switch between Hero and Stage selection” แสดงด้วยสีฟ้าอ่อน เพื่อแนะนำการสลับโหมดเลือก และข้อความ “Press ENTER to select hero” แสดงด้วยสีขาว บอกวิธีการยืนยันการเลือกตัวละคร

2.6.2.2 Selection Stage

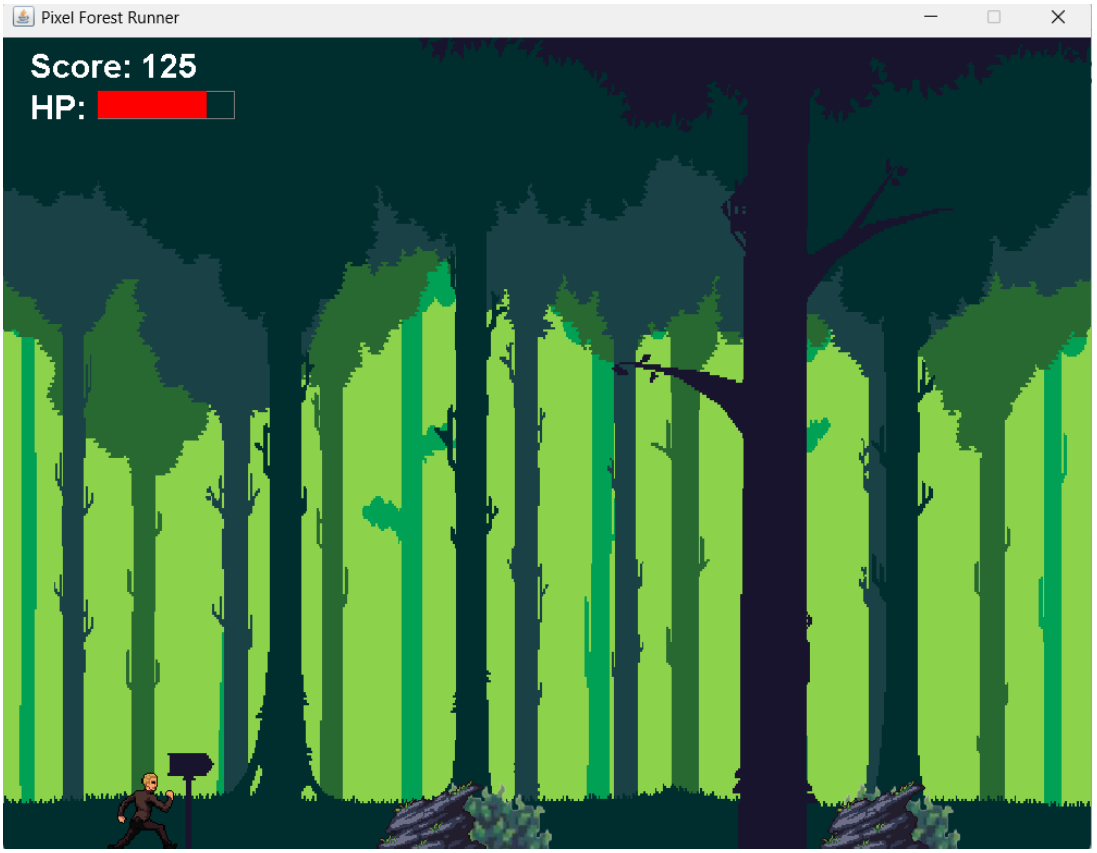


ประกอบไปด้วย

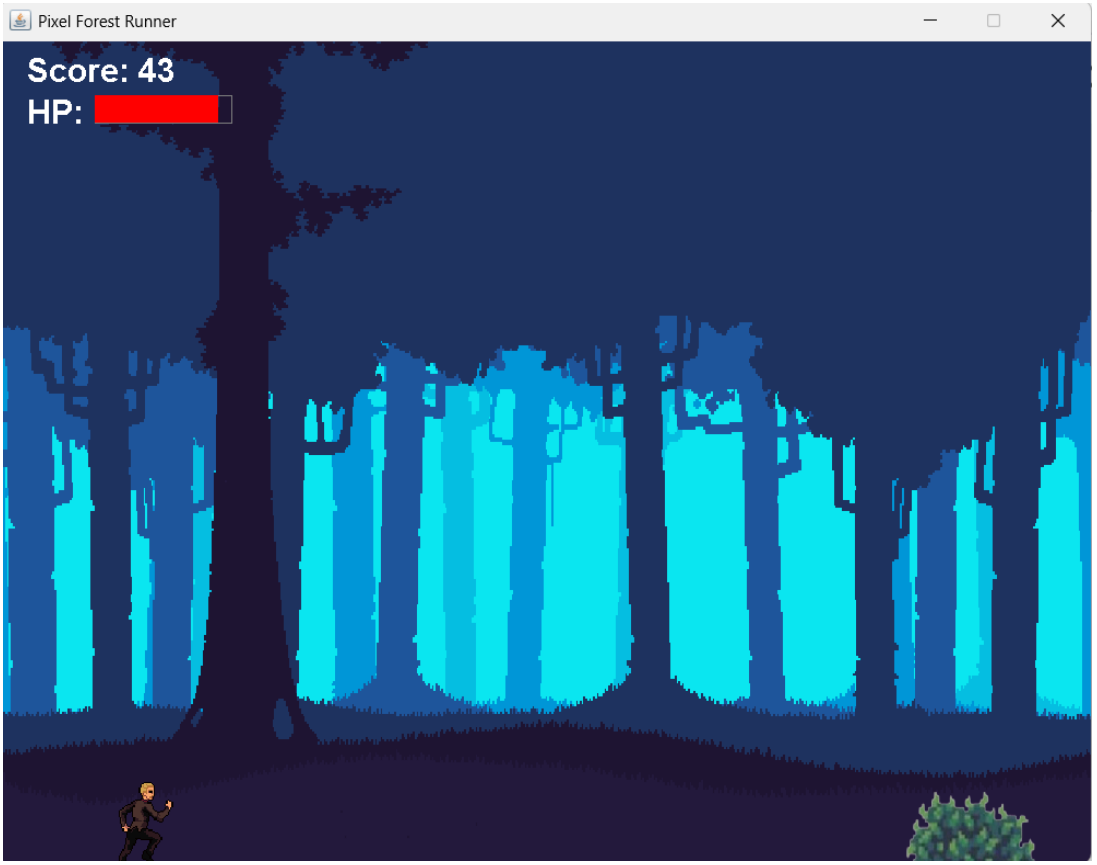
- Background Component (3 Component) - Black Background, The Forest of Dawn และ The Forest of Dusk
- Text Component (2 Component) - ข้อความ “Press TAB to switch between Hero and Stage selection” แสดงด้วยสีฟ้าอ่อน เพื่อแนะนำการสลับโหมดเลือก และข้อความ “Press ENTER to start game” แสดงด้วยสีขาว บอกวิธีการเริ่มเกม

2.6.3 หน้า Game

2.6.3.1 Stage The Forest of Dawn



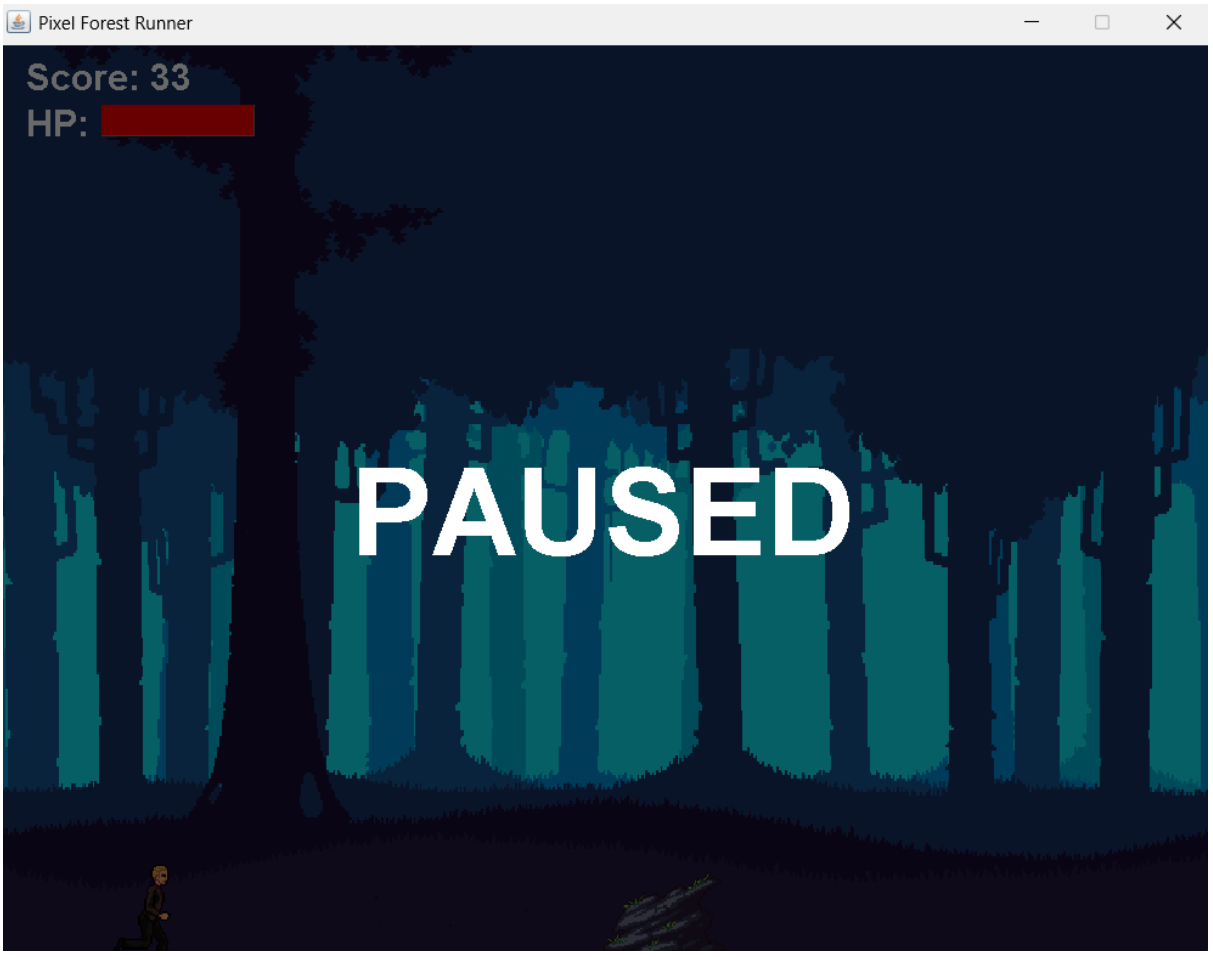
2.6.3.2 Stage The Forest of Dusk



ประกอบไปด้วย

- Background Component (3 Component) - Black Background, The Forest of Dawn และ The Forest of Dusk
- Text Component (2 Component) - Score Text ("Score: 43") แสดงคะแนนสะสมของผู้เล่นที่มุมซ้ายบนของหน้าจอ และ HP Bar Text ("HP:") แสดงค่าพลังชีวิตของผู้เล่น โดยมีแถบสีแดงเป็นตัวบ่งบอกปริมาณพลังชีวิตที่เหลืออยู่
- Person Component (3 Component) - Neo, Kenji (มีกรอบสีเหลืองแสดงว่ากำลังถูกเลือกอยู่) และ Boss

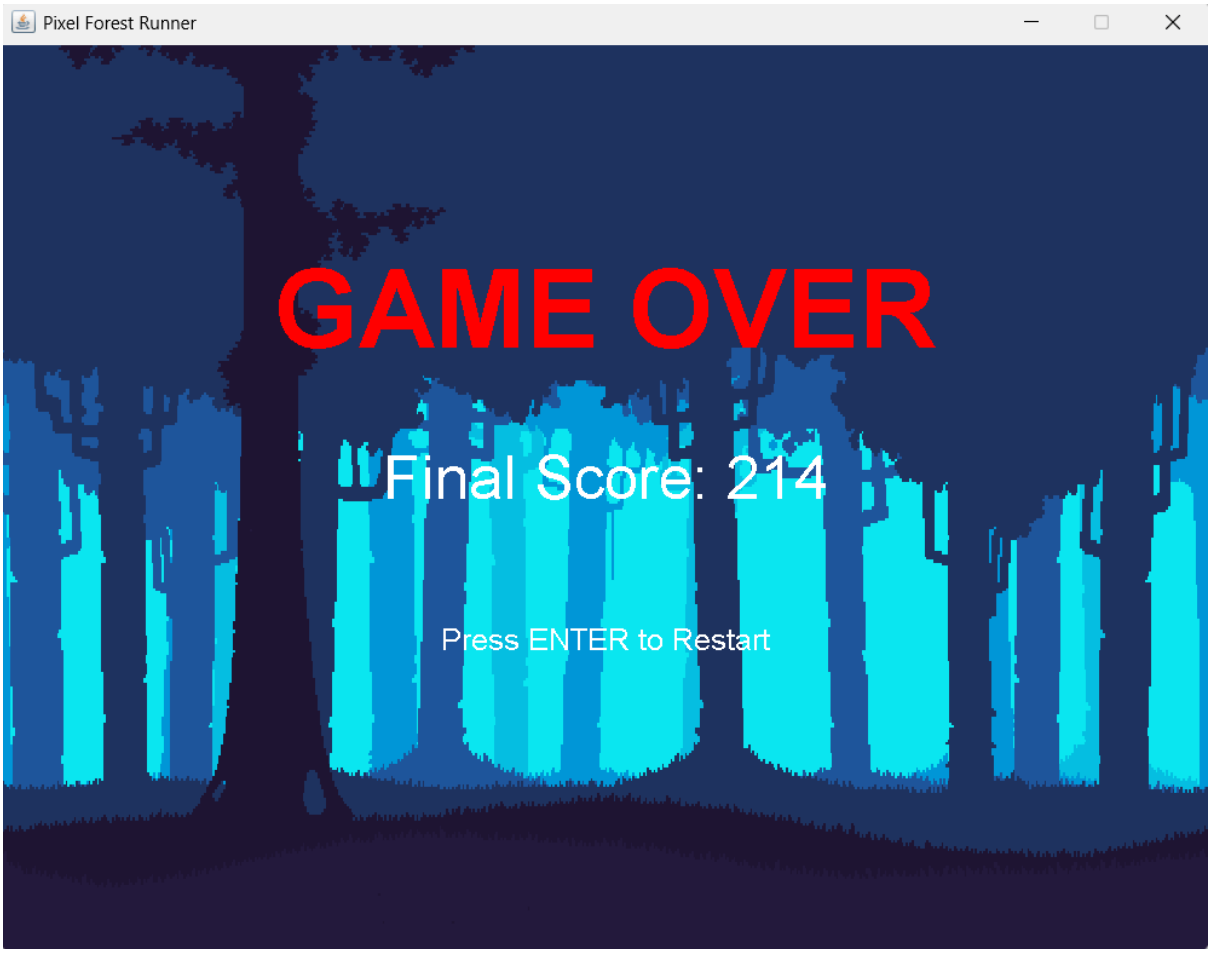
2.6.4 หน้า Pause



ประกอบไปด้วย

- Background Component (3 Component) - Black Transparent Overlay(ใช้โทนสีเข้มเพื่อเน้นข้อความ “PAUSED” ให้เด่นและลดความรบกวนของภาพพื้นหลัง), The Forest of Dawn และ The Forest of Dusk
- Text Component (1 Component) - Paused Text (“PAUSED”) ข้อความขนาดใหญ่ สีขาว อยู่กึ่งกลางหน้าจอ เพื่อให้ผู้เล่นรู้ว่าขณะนี้เกมอยู่ในโหมดหยุดพัก

2.6.5 หน้า Gameover



ประกอบไปด้วย

- Background Component (2 Component) - The Forest of Dawn และ The Forest of Dusk
- Text Component (3 Component) - Game Over Text (“GAME OVER”) เพื่อสื่อถึงความล้มเหลวของผู้เล่นอย่างชัดเจน, Final Score Text (“Final Score: 214”) แสดงคะแนนที่ผู้เล่นทำได้ในรอบนั้น เป็นการสรุปผลงานสุดท้าย และ Instruction Text (“Press ENTER to Restart”) เป็นข้อความแนะนำการกระทำต่อไป ให้ผู้เล่นสามารถเริ่มเกมใหม่ได้ทันที

2.7 Event Handling

```
public class Player extends GameObject {
    ...
    public void handleKeyPress(int keyCode) {
        if (keyCode == KeyEvent.VK_SPACE) {
            jump();
        }
    }
    public void handleKeyRelease(int keyCode) {}
    private void jump() {
        if (currentState != PlayerState.JUMPING) {
            verticalSpeed = jumpStrength;
            setState(PlayerState.JUMPING);
        }
    }
    @Override
    public void update() {
        if (y < groundLevel || verticalSpeed < 0) {
            verticalSpeed += gravity;
            y += verticalSpeed;
            setState(PlayerState.JUMPING);
        }
        if (y >= groundLevel && verticalSpeed > 0) {
            y = groundLevel;
```

handleKeyPress() เป็น Event Listener สำหรับคีย์บอร์ด (Jump)

update() ประมวลผล Physics Event และ State Event (วิ่ง/กระโดด)

setState() เปลี่ยน Animation ตาม Event

Hitbox ถูกอัปเดตทุกเฟรม → รองรับ Collision Event ทำการอัปเดตโดย method update()

verticalSpeed = 0;
setState(PlayerState.RUNNING);
}
int hitboxOffsetX = 34;
int hitboxWidth = 45;
int hitboxOffsetY = 45;
int hitboxHeight = 85;
updateHitbox(this.x + hitboxOffsetX, this.y + hitboxOffsetY, hitboxWidth, hitboxHeight);
}
...
}	

public class GamePanel extends JPanel implements Runnable {
...
public GamePanel() {
this.setPreferredSize(new Dimension(SCREEN_WIDTH, SCREEN_HEIGHT));
this.setFocusable(true);
this.currentGameState = new MenuState(this);
this.addKeyListener(new KeyboardInputs(this));
initClasses();
}
@Override
public void run() {
double drawInterval = 1_000_000_000.0 / FPS;
double delta = 0;
long lastTime = System.nanoTime();
long currentTime;
while (gameThread != null) {
currentTime = System.nanoTime();
delta += (currentTime - lastTime) / drawInterval;
lastTime = currentTime;
if (delta >= 1) {
update();
repaint();
delta--;
	KeyboardInputs เป็น Listener ที่จับเหตุการณ์ Key Press / Key Release จะส่งต่อไปยัง Player.handleKeyPress() หรือ handleKeyRelease()
	
	ใน method run() เป็น Game Loop (เรียก update และ draw ทุกเฟรม)

}
}
}
public void update() {
if (currentGameState != null) {
currentGameState.update();
}
}
...
}
	update() → ส่งต่อไปยัง GameState ปัจจุบัน
	update() จะเรียกทุก Object ที่อยู่ใน State ปัจจุบัน
	Player จะตอบสนองต่อ Key Event ผ่าน handleKeyPress() และ Physics Event ผ่าน update()

2.8 Algorithm ที่สำคัญในโปรแกรม

public abstract class GameObject {
protected int x, y;
protected int width, height;
protected Rectangle hitbox;
protected int hitboxOffsetX, hitboxOffsetY;
…
protected void updateHitbox() {
this.hitbox.x = this.x + this.hitboxOffsetX;
this.hitbox.y = this.y + this.hitboxOffsetY;
}
protected void updateHitbox(int x, int y, int width, int height) {
this.hitbox.x = x;
this.hitbox.y = y;
this.hitbox.width = width;
this.hitbox.height = height;
}
public Rectangle getHitbox() {
return this.hitbox;
}
protected void drawHitbox(Graphics g) {
g.setColor(Color.RED);
g.drawRect(hitbox.x, hitbox.y, hitbox.width, hitbox.height);
	ตัวแปร x และ y แสดงตำแหน่งของวัตถุในพื้นที่เกม ส่วน width และ height คือขนาดของวัตถุ ส่วน hitbox คือสี่เหลี่ยมที่ใช้ในการตรวจสอบการชน
	
	updateHitbox() จะอัปเดตตำแหน่งของ Hitbox โดยเพิ่มค่า hitboxOffsetX และ hitboxOffsetY ที่สามารถปรับเปลี่ยนตำแหน่งได้ตามต้องการ ซึ่งเป็นการทำให้ Hitbox สามารถมีการยืดหยุ่นตามการเคลื่อนที่ของวัตถุในเกม
	
	drawHitbox(Graphics g) ในตัวอย่างโค้ดนี้ จะทำให้

} ... }	กรอบสีแดงปรากฏขึ้นรอบๆ วัตถุ ใช้ในการ Debug

public class EntityManager {
private Player player;
private List<Obstacle> obstacles;
...
private void checkCollisions() {
if (!player.isAlive()) return;
Iterator<Obstacle> iterator = obstacles.iterator();
while (iterator.hasNext()) {
Obstacle obs = iterator.next();
if (player.getHitbox().intersects(obs.getHitbox())) {
player.takeDamage(10);
iterator.remove();
}
}
}
public void update() {
player.update();
Iterator<Obstacle> iterator = obstacles.iterator();
while (iterator.hasNext()) {
Obstacle obs = iterator.next();
obs.update();
}
checkCollisions();
}
public void draw(Graphics g) {
player.draw(g);
for (Obstacle obs : obstacles) {
obs.draw(g);
}
}
...
}
	ใน checkCollisions() มีการตรวจสอบสถานะของผู้เล่นก่อน ถ้าผู้เล่นตายแล้ว (isAlive() คืนค่า false) ก็ไม่จำเป็นต้องตรวจสอบการชนอีก วนลูปผ่านสิ่งกีดขวางทั้งหมด
	
	ตรวจสอบการชน เมธอด .intersects() ของคลาส Rectangle จะคืนค่า true ถ้าสองสี่เหลี่ยม มีพื้นที่ทับกัน player.getHitbox() คือกรอบสี่เหลี่ยม (Hitbox) ของผู้เล่น obs.getHitbox() คือกรอบสี่เหลี่ยมของสิ่งกีดขวาง
	
	ในเมธอด update() จะเห็นว่าเรียก checkCollisions() หลังจากอัปเดตการเคลื่อนไหวของทุกวัตถุแล้ว เกมจะอัปเดตตำแหน่งของผู้เล่นและสิ่งกีดขวางก่อน จากนั้นจึงตรวจสอบว่ามีการชนเกิดขึ้นในเฟรมปัจจุบันหรือไม่ หากชน จะมีการประมวลผลผลลัพธ์ เช่น ลดพลังชีวิต หรือลบวัตถุออกจากฉาก

บทที่ 3

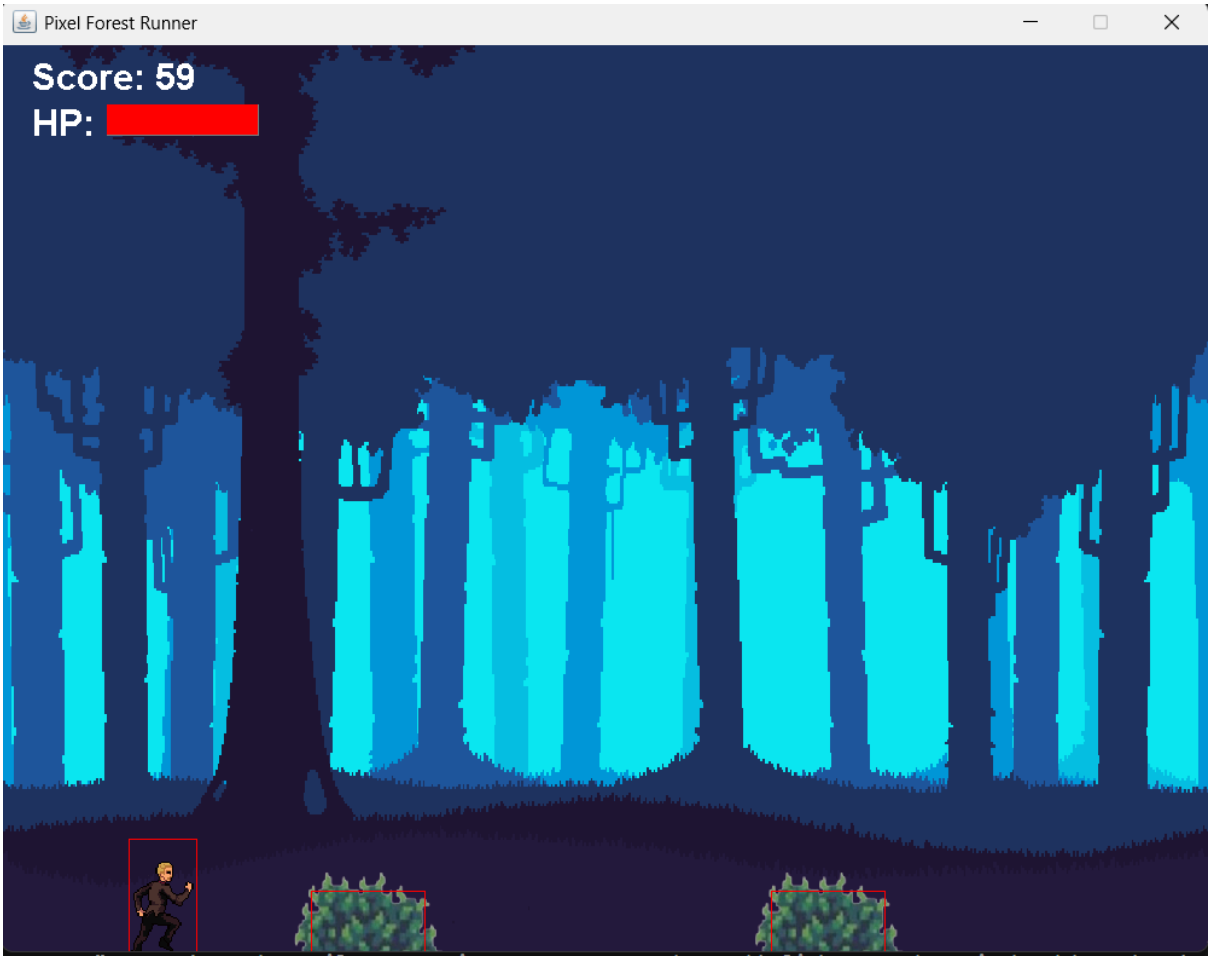
สรุป

3.1 ปัญหาที่พบ

3.1.1 ปัญหาด้าน Physics และ Gameplay

3.1.1.1 การกระโดดที่ไม่เป็นธรรมชาติ Player ใช้ฟิสิกส์แบบง่าย (เช่น การเพิ่ม/ลดค่า Y ในอัตราคงที่) แทนการใช้แรงโน้มถ่วง (Gravity) จริงๆ ทำให้การกระโดดดูไม่เหมือน Parabola (เส้นโค้งพาราโบลา) การแก้ไขต้องมีการใช้สูตรของนิวตันเพื่อจำลองฟิสิกส์ที่สมจริง

3.1.1.2 Hitbox ที่ไม่แม่นยำ Hitbox ของ Player และ Obstacle ถูกกำหนดเป็นสี่เหลี่ยมแบบง่าย(Bounding Box) ซึ่งจะใหญ่กว่าภาพจริงเล็กน้อย ทำให้ผู้เล่นรู้สึกว่าชนทั้งที่ยังไม่โดน



3.1.1.3 ความยากของเกมไม่สม่ำเสมอ ObstacleManager สร้างอุปสรรคแบบสุ่ม โดยไม่ได้พิจารณา จังหวะเวลา ระหว่างอุปสรรคที่ต่อเนื่องกัน (เช่น Rock ตามด้วย Bush ทันที) ทำให้เกิดสถานการณ์ที่ผู้เล่นไม่สามารถกระโดดหลบได้เลย

3.1.2 ปัญหาด้านการจัดการ Code และการขยายระบบ

3.1.2.1 การซ้ำซ้อนของโค้ดวาด/อัปเดต คลาสย่อยของ Obstacle เช่น Rock และ Bush มีโค้ดในเมธอด update() และ draw(g) ที่เหมือนกันเกือบทั้งหมด แต่จำเป็นต้อง Override แยกกัน

3.1.2.2 การจัดการ Score/Stat ที่ไม่เป็นระบบ คะแนน, สถิติ, และ High Score ถูกเก็บไว้กระจัดกระจาย (เช่น ใน PlayingState และ GameOverState) โดยไม่มีคลาสกลางอย่าง GameStats หรือ ScoreManager มาจัดการ

3.1.2.3 Memory Leak จาก Asset Loading ซ้ำ ทุกครั้งที่ SelectionState ถูกวาดใหม่, มันจะเรียก AssetLoader.loadStaticSprite(...) ซ้ำๆ ในลูป draw() ทำให้เกิดการโหลดรูปภาพซ้ำๆ ทุกเฟรม ซึ่งสิ้นเปลืองหน่วยความจำและ CPU

3.2 จดุเด่น

- โครงสร้างที่เอื้อต่อการทำ Multiplayer (Local)
    - Separation of Concerns: การแยก State ออกจาก GamePanel ทำให้ตรรกะของเกมไม่ปนกับ UI

- Entity-Based Design: การมีคลาส Player และ GameObject แยกจากกัน ทำให้การสร้างผู้เล่นคนที่สองทำได้ง่ายใน EntityManager
- Optimized Asset Loading: การรวมศูนย์การโหลดรูปภาพทั้งหมดไว้ที่คลาส AssetLoader ซึ่งใช้ ImageIO.read(InputStream) ช่วยลดข้อผิดพลาด Path: แก้ไขปัญหาการเข้าถึงไฟล์ที่ผิดพลาดได้ในที่เดียว และการใช้ InputStream ทำให้การโหลด Asset ยังคงทำงานได้แม้เกมจะถูก Build เป็น .jar ไฟล์แล้วก็ตาม

3.3 สรุปผลการพัฒนา

โครงการ PixelForestRunner มีวัตถุประสงค์เพื่อสร้างระบบเกมที่มีโครงสร้างครบถ้วนและสามารถขยายผลต่อยอดได้ในอนาคต การพัฒนาในครั้งนี้ได้ดำเนินการตามขอบเขตที่กำหนดไว้อย่างครบถ้วน ทั้งในด้านระบบหลัก การจัดการสถานะ (State Management) การรับเหตุการณ์ (Event Handling) การโหลดทรัพยากร (Asset Loading) ตลอดจนการแสดงผล (Rendering) ซึ่งทั้งหมดทำงานประสานกันอย่างมีประสิทธิภาพ

ในส่วนของ Core System ได้ออกแบบคลาส GamePanel ให้ทำหน้าที่เป็นศูนย์กลางในการควบคุมการทำงานของเกม โดยมี Game Loop ที่ทำงานได้อย่างเสถียรตามค่า FPS ที่กำหนด อีกทั้งยังสามารถสลับระหว่างสถานะของเกมได้อย่างราบรื่น พร้อมกันนี้ยังมีการใช้คลาส State แบบ Abstract เป็นโครงสร้างหลัก เพื่อแบ่งการทำงานของแต่ละฉากให้เป็นอิสระต่อกัน เช่น MenuState, SelectionState และ PlayingState ซึ่งช่วยให้โครงสร้างของเกมมีความเป็น