โครงงาน Project Mini Game

เรื่อง **เกมหุบเขาแห่งเวทมานตร์ (Mystic Valley)**

จัดทำโดย

นายณัทพล หาญจันทร์ 6704062612090

เสนอ

ผู้ช่วยศาสตราจารย์สถิตย์ ประสมพันธ์

โครงงานนี้เป็นส่วนหนึ่งของรายวิชา Object Oriented Programming

ภาควิชาวิทยาการคอมพิวเตอร์และสารสนเทศ คณะวิทยาศาสตร์ประยุกต์

มหาวิทยาลัยเทคโนโลยีพระจอมเหล้าพระนครเหนือ

ภาคเรียนที่ 1 ปีการศึกษา 2568

สารบัญ

บทที่ 1
บทนำ

1.1 เกี่ยวกับโครงงาน

ชื่อโครงงาน	: เกมหุบเขาแห่งเวทมานตร์ (Mystic Valley)
นำเสนอโดย	: นายณัทพล หาญจันทร์
อาจารย์ผู้สอน	: ผู้ช่วยศาสตราจารย์สถิตย์ ประสมพันธ์
Source Code	: https://github.com/nanonorth/Mystic_Valle

1.2 ที่มาและความสัมคัญ

โครงงานนี้จัดขึ้นเพื่อวัดผลการเรียนในรายวิชา Object Oriented Programming (OOP) โดยมีจุดประสงค์เพื่อให้นักศึกษาได้นำความรู้ที่ได้จากบทเรียนมาประยุกต์ใช้จริง ผ่านการสร้างผลงานในรูปแบบของเกม โดยการประยุกต์ใช้หลักการออกแบบเชิงวัตถุ (OOP) ในการพัฒนาเกมนี้ซึ่งช่วยเสริมความเข้าใจแนวคิดของการเขียนโปรแกรมเชิงวัตถุ

ในปัจจุบัน เทคโนโลยีด้านเกมคอมพิวเตอร์มีการพัฒนาอย่างต่อเนื่อง ทั้งในเชิงกราฟิก ระบบปัญญาประดิษฐ์ และกลไกการทำงานภายในเกม ซึ่งช่วยส่งเสริมทักษะการคิด วิเคราะห์ และการแก้ไขปัญหาของผู้เล่นได้เป็นอย่างดีโครงงาน “เกมหุบเขาแห่งเวทมนตร์ (Mystic Valley)” ได้รับการพัฒนาขึ้นเพื่อประยุกต์ความรู้ทางด้าน การเขียนโปรแกรมเชิงวัตถุ (Object-Oriented Programming : OOP) ในการสร้างสรรค์เกมขนาดเล็ก โดยมีเป้าหมายเพื่อให้ผู้จัดทำสามารถนำแนวคิดด้านการออกแบบเชิงวัตถุมาใช้กับการสร้างเกมจริง ทั้งในส่วนของการออกแบบโครงสร้างคลาส การจัดการวัตถุ การตรวจจับการชน (Collision Detection) การเคลื่อนไหวของตัวละคร และการแสดงผลภาพ (Rendering) ภายในสภาพแวดล้อมแบบ 2 มิติ ทั้งยังเสริมสร้างความสนุก ส่งเสริมสมาธิ และความเพลิดเพลินต่อผู้เล่น

1.3 ประเภทของโครงงาน

โครงงานพัฒนาเกมคอมพิวเตอร์ (Computer Game Development Project)

1.4 ประโยชน์

- ฝึกสมาธิ ไหวพริบ การหลบหลีก
- ฝึกการวางแผนในการกดโจมตี เพราะมีโอกาสพลาดและโดนโจมตีต่อเนื่องได้
- เพื่อความท้าทาย ความเพลิดเพลินและคลายเครียด

1.5 ขอบเขตของโครงงาน

ลำดับ	รายการ	5 – 10 (ตุลาคม)	11 – 20 (ตุลาคม)	20 – 25 (ตุลาคม)	25 – 31 (ตุลาคม)
1	ศึกษาข้อมูลและสร้างแผนที่				
2	ค้นหารูปตัวละและกราฟิกต่าง ๆ				
3	พัฒนาและเขียนโปรแกรม				
4	จัดทำเอกสารรายงาน				
5	ตรวจสอบและแก้ไขข้อผิดพลาด				

Proposal

ณัทพล หาญจันทร์ 6704062612090

Mystic Valley

- **รายละเอียดเกม**

  เกมเป็นแนว RPG + Life Simulator ที่ผู้เล่นจะได้รับบทเป็นนักผจญภัยในโลกเวทมนตร์ ผู้เล่นสามารถทำกิจกรรมต่าง ๆ ในชีวิตประจำวันได้ เช่น พูดคุยกับ NPC, ปลูกผักและยังสามารถเข้าสู่ดันเจี้ยนผ่านประตูเวทมนตร์ (Warp Gate) เพื่อสู้กับมอนสเตอร์และเก็บทรัพยากรที่มีค่าได้

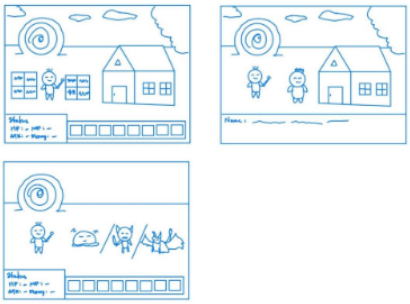
- **วิธีเล่น**

  ใช้ปุ่ม W/A/S/D ในการควบคุมการเดินในแผนที่ ใช้ mouse ในการคลิกเลือกช่องดินเพื่อปลูกผัเลือก NPC เพื่อพูดคุยหรือคลิก “Attack”

- **แมพ (Maps)**
  - Normal Map (Village/Farm)
    - ผู้เล่นสามารถปลูกผัก, พูดคุยกับ NPC
    - มี Warp Gate เพื่อเข้าสู่ดันเจี้ยน
  - Dungeon Map
    - ผู้เล่นสามารถสู้กับมอนสเตอร์ได้
    - มีระบบสุ่มมอนสเตอร์ (เช่น Slime, Goblin, Dragon Boss)
    - ผู้เล่นเลือกได้ว่าจะสู้ต่อหรือออกจากดันเจี้ยน

- **ระบบเกมเพลย์ (Gameplay Systems)**
  - Player System: ตัวละครผู้เล่น มี HP, MP, พลังโจมตี, Inventory, เงิน
  - NPC System: NPC ที่สามารถพูดคุยและโต้ตอบได้ (อาจเป็นร้านค้า/ให้เควส)
  - Farming System: ปลูก, รดน้ำ, เก็บเกี่ยวพืช
  - Battle System: ต่อสู้กับมอนสเตอร์ (โจมตีปกติ / ใช้เวทมนตร์)
  - Dungeon System: ระบบวนลูปสุ่มมอนสเตอร์ให้ผู้เล่นต่อสู้เรื่อย ๆ

- **ฉาก**



- **ประโยชน์**
  - ฝึกไหวพริบ ความว่องไว การหลบหลีกศัตรู
  - คลายเครียด

- **ตารางแผนการทำงาน**

ลำดับ	รายการ	1-10/09	11-15/09	16/09-5/10	6-7/10	10-13/10
1	สร้างแผนที่และออกแบบตัวละคร					
2	ออกแบบโครงสร้างตามหลัก OOP					
3	เขียนโค้ด					
4	จัดทำเอกสาร					
5	ตรวจสอบและแก้ไข					

บทที่ 2
ส่วนการพัฒนา

2.1 รายละเอียดเกม

เกม “หุบเขาแห่งเวทมนตร์ (Mystic Valley)” เป็นเกมแนว 2 มิติ (2D Adventure Game) ที่ผู้เล่นจะได้รับบทเป็นพ่อมดในหุบเขาแห่งมนตรา ซึ่งเต็มไปด้วยอันตรายและพลังเวทลึกลับ ผู้เล่นต้องต่อสู้กับศัตรูและเอาชนะบอสประจำดันเจี้ยนเพื่อปกป้องหมู่บ้านจากพลังชั่วร้ายที่กำลังรุกราน เกมถูกออกแบบให้มีรูปแบบการเล่นที่เข้าใจง่าย แต่ยังคงความท้าทาย โดยใช้ระบบพื้นฐานของเกมผจญภัย เช่น การเคลื่อนไหวของตัวละคร การโจมตี การหลบหลีก การตรวจจับการชน (Collision Detection) และระบบการฟื้นฟูค่าพลังชีวิต (HP), มานา (MP) และสตามินา (Stamina)

เกมถูกออกแบบให้มีสไตล์ภาพแบบพิกเซลอาร์ต (Pixel Art) ให้ความรู้สึกคลาสสิกและเล่นได้อย่างลื่นไหล ด้วยระบบกลไกที่จำลองลักษณะของเกมเชิงพาณิชย์ในระดับเบื้องต้น
โดยอาศัยแนวคิดการเขียนโปรแกรมเชิงวัตถุ (Object-Oriented Programming) ในการจัดการวัตถุทุกประเภทในเกม เช่น ตัวละคร แผนที่ ไฟล์ภาพ และการตรวจจับการชน (Collision)

ระบบหลักของเกมประกอบด้วย

1. ระบบตัวละคร (Character System)
   - ผู้เล่นมีค่าสถานะต่าง ๆ เช่น พลังชีวิต ความเร็วและพลังโจมตี ซึ่งจะปรับเปลี่ยนตามระดับพลังเลเวลของผู้เล่น
2. ระบบบอส (Boss System)
   - ศัตรูหลักในเกมที่มีรูปแบบการโจมตีหลายรูปแบบ เช่น การโจมตีระยะประชิด
3. ระบบแผนที่ (Map System)
   - แบ่งเป็น 2 พื้นที่หลัก ได้แก่ หมู่บ้าน (Village) และดันเจี้ยน (Dungeon) โดยผู้เล่นสามารถวาร์ประหว่างพื้นที่ได้
4. ระบบการเกิดใหม่ (Respawn System)
   - เมื่อผู้เล่นพ่ายแพ้ จะกลับไปเกิดใหม่ที่หมู่บ้าน พร้อมรีเซ็ตค่าพลังตามระดับเลเวล
5. ระบบอินเทอร์เฟซ (User Interface) – แสดงค่าพลัง HP, MP, Stamina รวมถึงแถบพลังของบอส

2.2 เนื้อเรื่องของเกม

ในหุบเขาอันเงียบสงบที่ชื่อว่า Mystic Valley ซึ่งเคยเป็นแหล่งรวมของเหล่าผู้ใช้เวทมนตร์และสิ่งมีชีวิตวิเศษ ได้เกิดเหตุไม่คาดฝันขึ้นเมื่อพลังมืดที่ถูกผนึกไว้ในดันเจี้ยนใต้หุบเขาเริ่มตื่นขึ้นอีกครั้ง “จิ้งจอกเก้าหาง Kitsuna” มารร้ายแห่งพลังเวทได้ฟื้นคืนชีพและเริ่มทำลายความสงบสุขของหมู่บ้าน

ผู้เล่นได้รับบทเป็น “นักเวทหนุ่มผู้ถูกเลือก” ผู้ต้องออกเดินทางไปยังดันเจี้ยน เพื่อต่อสู้และผนึกพลังมืดให้กลับคืนสู่ความสงบอีกครั้ง ระหว่างทางผู้เล่นจะต้องฝ่าฝันอุปสรรคมากมาย ฝึกฝนความสามารถทางเวทมนตร์ และใช้กลยุทธ์ต่อสู้เพื่อเอาชนะศัตรู

2.3 วิธีการเล่นเกม

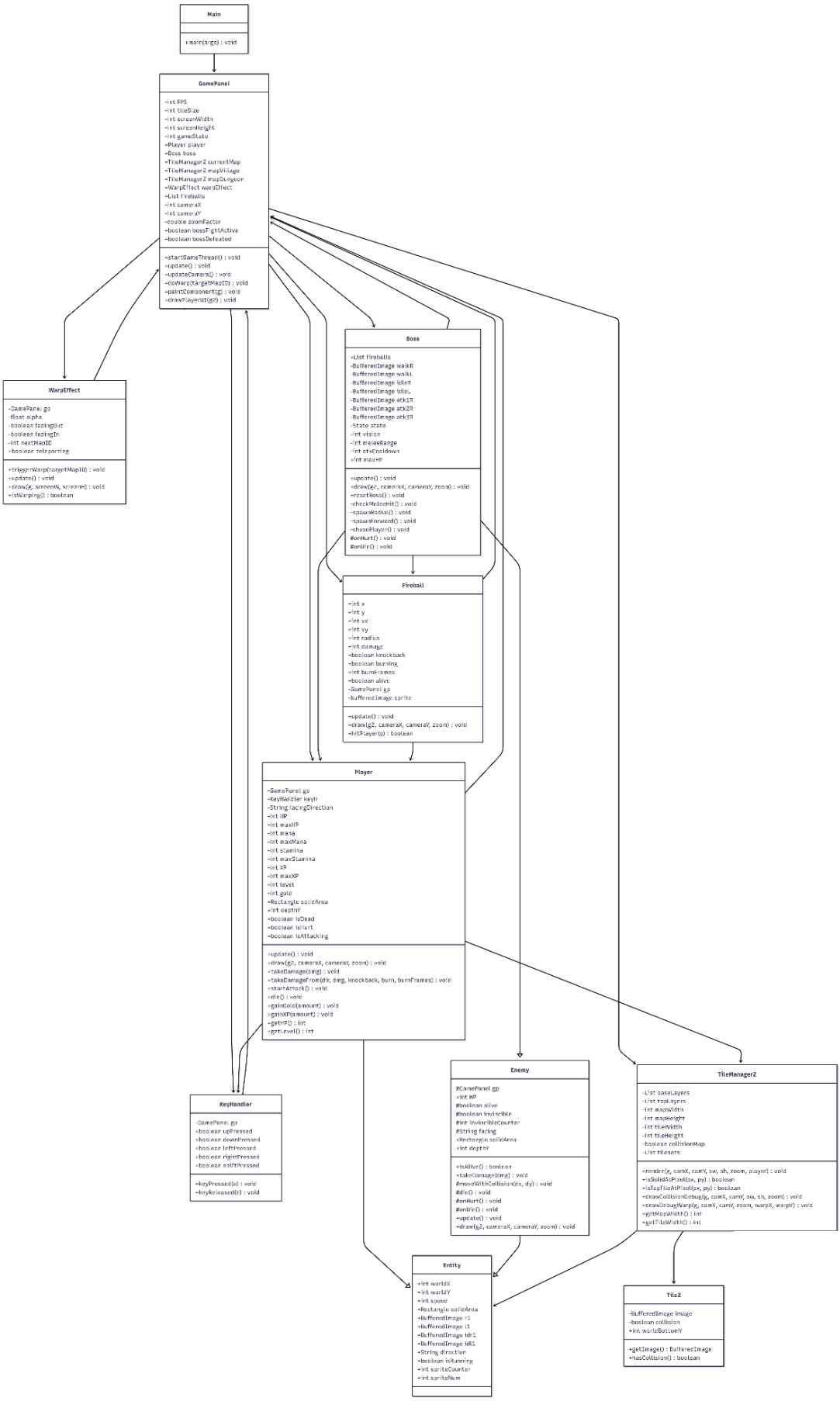
เกม Mystic Valley ถูกออกแบบให้ควบคุมได้ง่ายและเหมาะสำหรับผู้เล่นทุกระดับ โดยใช้คีย์บอร์ดและเมาส์ในการควบคุมหลัก ดังนี้

- ใช้ W A S D เคลื่อนไหวขึ้น / ซ้าย / ลง / ขวา
- ใช้ SHIFT วิ่งเร็วขึ้น (ใช้สตามินา)
- ใช้ Mouse โจมตีศัตรู

เงื่อนไขการเล่นต่าง ๆ ในเกม

- เมื่อค่าพลังชีวิต (HP) ลดลงจนหมด ตัวละครจะเสียชีวิตและกลับไปเกิดใหม่ที่หมู่บ้าน
- ค่ามานา (MP) และสตามินา (Stamina) จะค่อย ๆ ฟื้นฟูตามเวลาหรือเมื่อผู้เล่นพัก
- เมื่อผู้เล่นสามารถเอาชนะศัตรูได้จะได้รับเงินเลเวลและเงิน โดยความยากของเกมจะถูกคิดตาม เลเวลของผู้เล่น เช่น ศัตรูโจมตีแรงขึ้น เลือดมากขึ้น เป็นต้น
- โดยจะถือว่าชนะเกมได้เมื่อผู้เล่นเลเวลถึง 30 จากนั้นจะทำการเริ่มต้นใหม่ทุกอย่าง

2.4 Class Diagram



Class Diagram ของ 11 คลาสหลัก แบ่งออกเป็น 3 หมวดหมู่

1) Application Layer

1.1) Main Class: จุดเริ่มต้นโปรแกรม

1.2) GamePanel Class: Game Loop และการแสดงผล

1.3) KeyHandler Class: การจัดการ Input

1.4) WarpEffect Class: Visual Effects สำหรับการเปลี่ยนแผนที่

2) Entity System

2.1) Entity: Base Class สำหรับตัวละครทั้งหมด

2.2) Player Class: ตัวละครหลักที่ผู้เล่นควบคุม

2.3) Enemy Class: Abstract Class สำหรับศัตรู

2.4) Boss Class: ศัตรูบอสที่มี AI ซับซ้อน

2.5) Fireball Class: Projectile System

3) Tile System

3.1) TileManager2 Class: การจัดการแผนที่

3.2) Tile2 Class: Tile แต่ละช่อง

การออกแบบและสถาปัตยกรรมระบบ

Class Diagram และความสัมพันธ์ระหว่างคลาส

จาก Class Diagram ระบบประกอบด้วยคลาสหลัก 11 คลาส แบ่งเป็น 3 Package

```
Mystic_Valley/
├── main/
│   ├── Main.java
│   ├── GamePanel.java
│   ├── KeyHandler.java
│   └── WarpEffect.java
├── entity/
│   ├── Entity.java
│   ├── Player.java
│   ├── Enemy.java
│   ├── Boss.java
│   └── Fireball.java
└── tile/
    ├── TileManager2.java
    └── Tile2.java
```

ความสัมพันธ์หลัก (Relationships)

1) Composition (◆) - Has-a แบบแน่นแฟ้น:

1.1) GamePanel ◆ Player

- GamePanel เป็นเจ้าของ Player และควบคุมวงจรชีวิตของ Player

1.2) GamePanel ◆ Boss

- GamePanel สร้างและจัดการ Boss

1.3) GamePanel ◆ KeyHandler

- Input Handler เป็นส่วนหนึ่งของ GamePanel

1.4) GamePanel ◆ WarpEffect

-Effect System อยู่ภายใต้ GamePanel

2) Aggregation (◇) - Has-a แบบหลวม:

2.1) GamePanel ◇ TileManager2

- GamePanel ใช้งาน TileManager2 แต่ไม่ได้เป็นเจ้าของโดยตรง

2.2) GamePanel ◇ Fireball

- GamePanel จัดการ List ของ Fireball แต่ Boss เป็นผู้สร้าง

2.3) TileManager2 ◇ Tile2

- TileManager2 จัดการ Tile หลายๆ ชั้น

3) Inheritance (◁—) - Is-a:

3.1) Entity ◁— Player

- Player เป็น Entity

3.2) Entity ◁— Enemy

- Enemy เป็น Entity

3.3) Enemy ◁— Boss

- Boss เป็น Enemy

4) Dependency (•••>) - Uses:

4.1) Player •••> KeyHandler

- Player อ่านค่าจาก KeyHandler

4.2) Player •••> TileManager2

- Player ตรวจสอบ Collision กับแผนที่

4.3) Boss •••> Player

- Boss ตรวจจับตำแหน่งและโจมตี Player

4.4) Fireball •••> Player

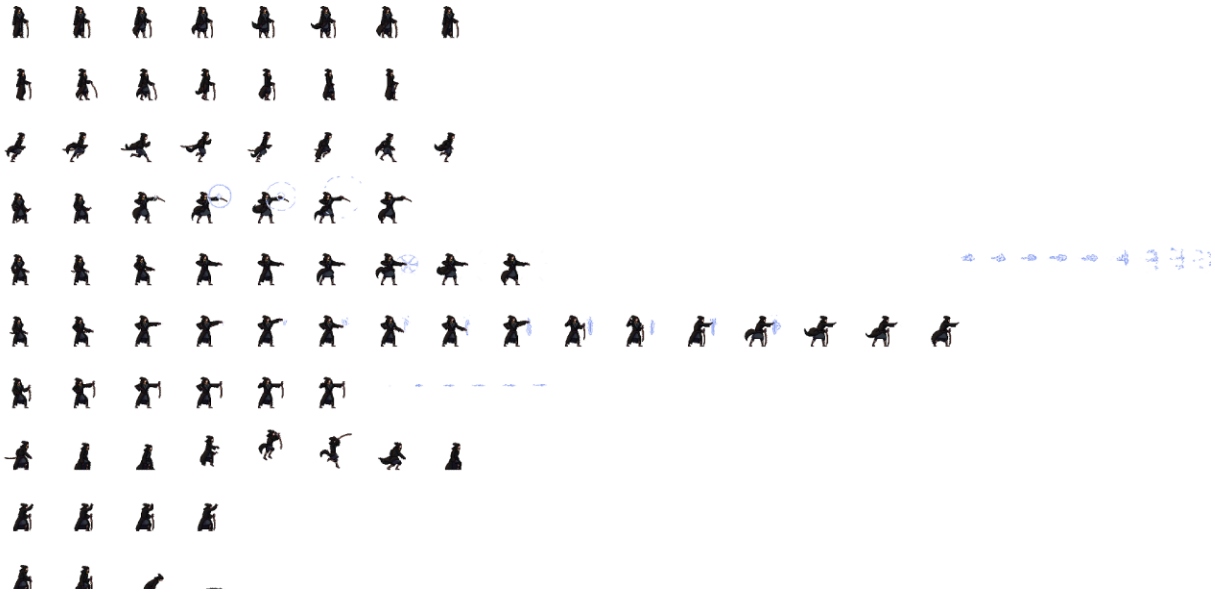
- Fireball ตรวจสอบการชนกับ Player

2.5 รูปแบบการพัฒนา

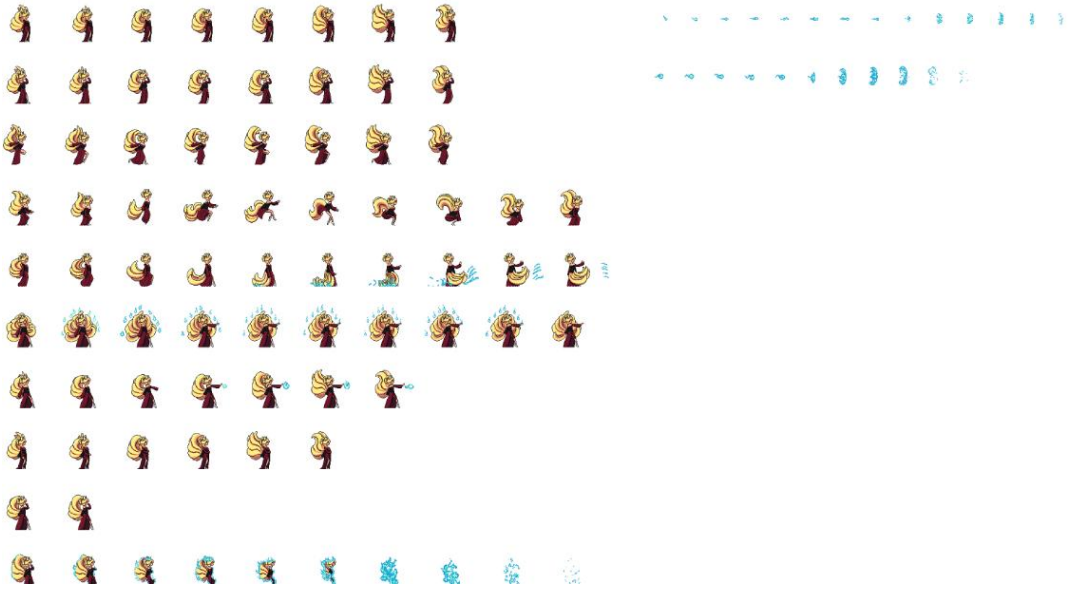
- ภาษา: Java IDE: Eclipse / IntelliJ IDEA
- Library: Java AWT, Swing (JFrame, JPanel, Graphics2D)
- เครื่องมือจัดการภาพ: Tiled Map Editor
- เครื่องมือออกแบบภาพ: Photoshop / Aseprite
- ระบบควบคุมเวอร์ชัน: GitHub

2.6 Assets Credit

Player: Free Pixel Wizard 2D Art (https://free-game-assets.itch.io/free-wizard-sprite-sheets-pixel-art)

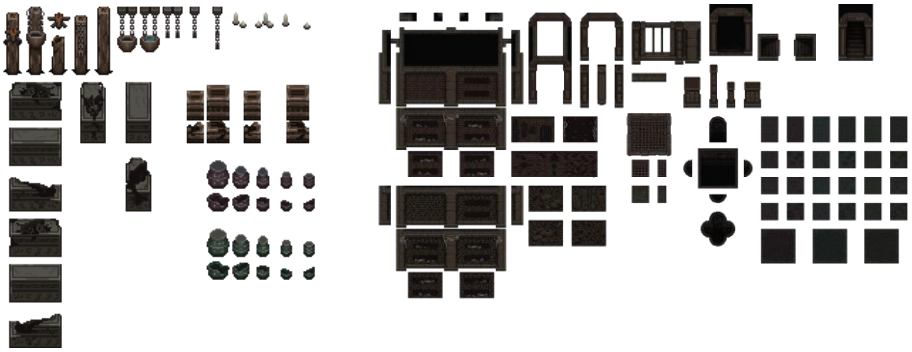


Boss: Free Yokai Pixel Art Character Sprites (https://craftpix.net/freebies/free-yokai-pixel-art-character-sprites/)



Map Valley: Pixel Asset Pack Jam #5 | Winter (https://wurls.itch.io/snowy-asset-pack)

Map Dungeon: Rogue Fantasy Catacombs (https://szadiart.itch.io/rogue-fantasy-catacombs)



2.7 แนวคิดการเขียนโปรแกรมเชิงวัตถุ

2.7.1 Constructor

GamePanel Class

```
public GamePanel() {
    setPreferredSize(new Dimension(screenWidth, screenHeight));
    setBackground(Color.black);
    setDoubleBuffered(true);
    keyH = new KeyHandler(this);
    addKeyListener(keyH);
    setFocusable(true);
    mapVillage = new TileManager2("res/map/Map_V1.tmj");
    mapDungeon = new TileManager2("res/map/Map_D1.tmj");

    try {
        pauseBG = ImageIO.read(new File("res/UI/CHAT1.png"));
    } catch (Exception e) {
    }
    currentMap = mapVillage;
    currentMapId = 0;
    player = new Player(this, keyH);
    player.setSpawnPoint(warpXS, warpYS);

    addMouseListener(new java.awt.event.MouseAdapter() {
        @Override
        public void mousePressed(java.awt.event.MouseEvent e) {
            if (!player.isAttacking && !player.isDead) {
                player.startAttack();
            }
        }
    });
}
```

เป็น Constructor ของคลาส GamePanel ใช้สำหรับกำหนดค่าเริ่มต้นของเกม เช่น ตั้งขนาดจอเกม โหลดแผนที่ (Map_V1, Map_D1), สร้าง Player, Boss, KeyHandler, ผูก Event เมาส์ (โจมตีเมื่อคลิก), 0 เป็นจุดเริ่มต้นของการสร้างวัตถุเกมก่อนรัน Thread

Tile2 Class

```
public Tile2(BufferedImage image, boolean collision) {
    this.image = image;
    this.collision = collision;
}
```

ใช้สร้าง **วัตถุ (Object)** ของ Tile2 แต่ละช่องบนแผนที่ กำหนดค่าตั้งต้นของ tile เช่น รูปภาพ (image) และการชน (collision) เมื่อโหลดแมพจากไฟล์ (เช่น TMJ หรือ JSON) จะมีการเรียก constructor นี้หลายครั้งเพื่อสร้าง tile แต่ละช่องใน world

TileManager2 Class

```
public TileManager2(String mapPath) {
    try {
        String jsonText = new String(Files.readAllBytes(Paths.get(mapPath)));
        JSONObject json = new JSONObject(jsonText);

        mapWidth  = json.getInt("width");
        mapHeight = json.getInt("height");
        tileWidth = json.getInt("tilewidth");
        tileHeight= json.getInt("tileheight");

        collisionMap = new boolean[mapHeight][mapWidth];

        loadTilesets(json, mapPath);
        loadLayers(json);
    } catch (Exception e) { e.printStackTrace(); }
}
```

ทำหน้าที่โหลดข้อมูลแผนที่จากไฟล์ .json หรือ .tmj เมื่อสร้างอ็อบเจกต์ TileManager เป็นตัวเริ่มต้น (Initialization) ของข้อมูลสำคัญ เช่น mapWidth, tileWidth, collisionMap เป็น *Constructor* ที่เตรียมพร้อมข้อมูลให้พร้อมใช้งานตั้งแต่เริ่มต้นโปรแกรม

Player Class

```
public Player(GamePanel gp, KeyHandler keyH) {
    this.gp = gp;
    this.keyH = keyH;
    setDefaultValues();
    getPlayerImage();
}
```

หน้าที่สร้างวัตถุ Player ขึ้นมา พร้อมกำหนดค่าเริ่มต้น เช่น ความเร็ว, ทิศทาง, และโหลดภาพ Sprite ของตัวละคร เป็นตัวอย่างของการกำหนดสภาวะเริ่มต้นของ Object เมื่อถูกสร้าง (Initialization)

Enemy Class

```
public Enemy(GamePanel gp) {
    this.gp = gp;
    speed = 1;
}
```

เป็นตัว Constructor ของคลาส Enemy ใช้สำหรับกำหนดค่าเริ่มต้น เช่น ความเร็ว (speed = 1) และเชื่อมโยงกับ GamePanel ซึ่งเป็นองค์ประกอบหลักของเกม ทำให้เมื่อสร้างวัตถุศัตรู (เช่น Boss, Monster) ทุกตัวมีการเตรียมค่าพื้นฐานพร้อมใช้งานโดยอัตโนมัติ

Boss Class

```
public Boss(GamePanel gp, List<Fireball> sharedProjectiles) {
    super(gp);
    this.gpProjectiles = sharedProjectiles;

    this.maxHP = BASE_HP;
    this.HP = maxHP;

    solidArea = new Rectangle(40, 150, 70, 40);

    worldX = 20;
    worldY = 80;

    speed = 1;
    loadSprites();
}
```

ใช้สำหรับกำหนดค่าเริ่มต้นของบอส เช่น HP, ตำแหน่ง, ความเร็ว, พื้นที่ชน (solidArea) และโหลดภาพ มีการเรียก super(gp) เพื่อส่ง GamePanel ให้คลาสแม่ (Enemy) ⟶ แสดงการใช้ Inheritance ร่วมกับ Composition

WarpEffect Class

```
public WarpEffect(GamePanel gp) {
    this.gp = gp;
}
```

ใช้ในการ สร้างวัตถุ WarpEffect พร้อมรับพารามิเตอร์ GamePanel gp เพื่อเชื่อมโยงกับหน้าจอหลักของเกม เมื่อเรียก new WarpEffect(gp) ตัวแปร gp จะถูกเก็บไว้ในคลาสนี้ เพื่อให้เข้าถึงฟังก์ชันอื่น ๆ ของเกมได้ เช่น gp.doWarp(nextMapID)

KeyHandler Class

```
public KeyHandler(GamePanel gp) {
    this.gp = gp;
}
```

เป็น Constructor ใช้สำหรับสร้างอ็อบเจ็กต์ของ KeyHandler โดยรับพารามิเตอร์ GamePanel gp เพื่อเชื่อมโยงกับคลาสหลักของเกม ทำให้ KeyHandler สามารถเข้าถึงและควบคุมสถานะของเกม เช่น gp.gameState หรือ gp.player.takeDamage() ได้

2.7.2 Encapsulation (การห่อหุ้ม)

GamePanel Class

```
private void drawBar(Graphics2D g2, int x, int y, int width, int height, int current, int max, Color fillColor,
        Color bgColor, String label) {

    // Background
    g2.setColor(bgColor);
    g2.fillRoundRect(x, y, width, height, 8, 8);

    // Fill
    double percent = Math.max(0, Math.min(1, (double) current / max));
    int fillWidth = (int) (width * percent);
    g2.setColor(fillColor);
    g2.fillRoundRect(x, y, fillWidth, height, 8, 8);

    // Border
    g2.setColor(Color.WHITE);
    g2.drawRoundRect(x, y, width, height, 8, 8);

    // Center Text
    g2.setFont(g2.getFont().deriveFont(Font.BOLD, 12f));
    String text = label + ": " + current + "/" + max;
    FontMetrics fm = g2.getFontMetrics();

    int textX = x + (width - fm.stringWidth(text)) / 2;
    int textY = y + ((height - fm.getHeight()) / 2) + fm.getAscent();
```

เมธอด drawBar() ถูกกำหนดเป็น private ⟶ ป้องกันไม่ให้ถูกเรียกจากภายนอกคลาส ใช้ภายใน drawPlayerUI() เพื่อแสดงหลอดพลัง (HP, Mana, Stamina, XP) เป็นการซ่อนรายละเอียดและให้เรียกผ่าน method ที่จัดการเฉพาะได้อย่างปลอดภัย

Tile2 Class

```
    private int mapWidth, mapHeight, tileWidth, tileHeight;
    private boolean[][] collisionMap;
```

ตัวแปร image และ collision ถูกประกาศเป็น private ไม่ให้คลาสอื่นแก้ไขโดยตรง การเข้าถึงทำผ่าน getter method ช่วยป้องกันไม่ให้ส่วนอื่นในเกมแก้ไขข้อมูลภายในโดยไม่ตั้งใจ

TileManager2 Class

```
private final ArrayList<Tile2[][]> baseLayers = new ArrayList<>();
private final ArrayList<Tile2[][]> topLayers  = new ArrayList<>();
private int mapWidth, mapHeight, tileWidth, tileHeight;
private boolean[][] collisionMap;
```

ตัวแปรทั้งหมดถูกประกาศเป็น private เพื่อ ป้องกันการเข้าถึงโดยตรงจากภายนอก มี Getter ใช้เข้าถึงข้อมูลอย่างปลอดภัย

```
public int getMapWidth(){ return mapWidth; }
public int getMapHeight(){ return mapHeight; }
public int getTileWidth(){ return tileWidth; }
public int getTileHeight(){ return tileHeight; }
```

Player Class

```
private int HP = 100;
private int maxHP = 100;
private int baseHP = 100;

private int mana = 100;
private int maxMana = 100;
private int baseMana = 100;

private int stamina = 100;
private int maxStamina = 100;
private int baseStamina = 100;
private boolean staminaDepleted = false;

private int XP = 0;
private int maxXP = 100;

private int baseXP = 0;
private int basemaxXP = 100;


private int level = 1;

private int gold = 0;
```

```
public int getMaxHP() {
    return maxHP;
}

public int getMana() {
    return mana;
}

public int getMaxMana() {
    return maxMana;
}

public int getStamina() {
    return stamina;
}

public int getMaxStamina() {
    return maxStamina;
}
```

Enemy Class

```
protected boolean alive = true;
protected boolean invincible = false;
protected int invincibleCounter = 0;
```

Boss Class

```
private BufferedImage[] hurtL = new BufferedImage[2];
private BufferedImage[] deadR = new BufferedImage[10];
private BufferedImage[] deadL = new BufferedImage[10];
private BufferedImage[] atk1R = new BufferedImage[10];
private BufferedImage[] atk1L = new BufferedImage[10];
private BufferedImage[] atk2R = new BufferedImage[10];
private BufferedImage[] atk2L = new BufferedImage[10];
private BufferedImage[] atk3R = new BufferedImage[7];
private BufferedImage[] atk3L = new BufferedImage[7];


// Encapsulation

private int frameCounter = 0;
private int frameIndex = 0;

private enum State { IDLE, WALK, ATK1, ATK2, ATK3, HURT, DEAD }
private State state = State.IDLE;

private static final int BASE_HP = 60;

private int vision = 360;
private int meleeRange = 60;
private int damageBoost = 0;

private int atkCooldown = 0;
private int atkInterval = 90; // frames between attacks when in range
```

WarpEffect Class

```
private GamePanel gp;

private float alpha = 0f;
private boolean fadingOut = false;
private boolean fadingIn = false;
private int nextMapID;
```

2.7.3 Composition (การประกอบวัตถุ)

GamePanel Class

```
public TileManager2 mapVillage;
public TileManager2 mapDungeon;
public TileManager2 currentMap;

public int currentMapId = 0;

public Player player;

public Boss boss;

public WarpEffect warpEffect = new WarpEffect(this);
```

คลาส GamePanel ประกอบด้วยวัตถุของคลาสอื่นๆ การออกแบบแบบนี้คือ Composition เพราะ GamePanel จะถือครองและจัดการอายุการทำงานของอ็อบเจกต์เหล่านี้

TileManager2 Class

```
private final ArrayList<Tile2[][]> baseLayers = new ArrayList<>();
private final ArrayList<Tile2[][]> topLayers  = new ArrayList<>();
```

TileManager2 ประกอบด้วยอ็อบเจกต์ Tile2 หลายตัวในรูปแบบตาราง ใช้ คลาสภายใน Tileset เพื่อเก็บข้อมูล tileset เฉพาะแต่ละแผนที่ แสดงถึงความสัมพันธ์แบบ “has-a” เช่น

Player Class

```
GamePanel gp;
KeyHandler keyH;
```

Player ประกอบด้วยอ็อบเจกต์จากคลาสอื่น เช่น GamePanel ใช้ควบคุมภาพรวมของเกม KeyHandler ใช้รับเหตุการณ์จากคีย์บอร์ด

Enemy Class

```
protected final GamePanel gp;
```

Boss Class

```
public ArrayList<Fireball> fireballs = new ArrayList<>();
```

WarpEffect Class

```
private GamePanel gp;
```

2.7.4 Inheritance (การสืบทอด)

Player Class

```
public class Player extends Entity {

    GamePanel gp;
    KeyHandler keyH;
    private String facingDirection = "right";

    public Rectangle solidArea = new Rectangle(10, 200, 30, 20);
```

หน้าที่สืบทอดตัวแปรพื้นฐาน (เช่น worldX, worldY, speed, BufferedImage) และ method จาก Entity ช่วยลดการเขียนโค้ดซ้ำ และเพิ่มความสามารถเฉพาะตัวใน Player เช่น ระบบ HP, XP, การโจมตี ฯลฯ

Enemy Class

```
public abstract class Enemy extends Entity {

    protected final GamePanel gp;

    // Basic state
    public int HP = 200;
    protected boolean alive = true;
    protected boolean invincible = false;
    protected int invincibleCounter = 0;
    protected int invincibleTime = 30;
```

Enemy สืบทอดจาก Entity (ซึ่งเก็บข้อมูลพื้นฐาน เช่น worldX, worldY, speed, sprite ฯลฯ) ช่วยให้ Enemy ไม่ต้องเขียนซ้ำในทุกคลาส เช่น Boss หรือ Monster ก็แค่ extends Enemy

Boss Class

```
public class Boss extends Enemy {
```

Boss สืบทอดคุณสมบัติจาก Enemy ซึ่งต่อยอดจาก Entity อีกชั้น ทำให้คลาส Boss ได้รับคุณสมบัติพื้นฐาน เช่น worldX, worldY, HPและเมธอดการเคลื่อนไหวโดยอัตโนมัติเพิ่มพฤติกรรมเฉพาะของตนเอง เช่น spawnForward(), spawnRadial(), checkMeleeHit()

2.7.5 Abstraction

TileManager2 Class

```
public void render(Graphics2D g, int camX, int camY, int sw, int sh, double zoom, Player player) {

    // Base
    for (Tile2[][] layer : baseLayers) {
        for (int y=0; y<mapHeight; y++)
            for (int x=0; x<mapWidth; x++)
                if (layer[y][x]!=null) drawTile(g, layer[y][x], x, y, camX, camY, sw, sh, zoom);
    }

    // Buckets + player depth
    List<RenderBucket> bucket = new ArrayList<>();

    for (Tile2[][] layer : topLayers) {
        for (int y=0; y<mapHeight; y++) {
            for (int x=0; x<mapWidth; x++) {
                Tile2 t = layer[y][x]; if (t==null) continue;
                final int tx=x, ty=y; final Tile2 tt=t;
                bucket.add(new RenderBucket(){{
```

ซ่อนรายละเอียดการวาดและคำนวณทั้งหมดไว้ในฟังก์ชันเดียว ผู้ใช้คลาสไม่จำเป็นต้องรู้ว่า tile ถูกโหลดและคำนวณอย่างไร เพียงแค่เรียก render() ก็จะแสดงผลแผนที่ได้ครบ

Player Class

```
public void draw(Graphics2D g2, int cameraX, int cameraY, double zoom) {
    BufferedImage image = null;
```

ซ่อนกระบวนการวาด sprite ที่ซับซ้อนจากภายนอก ภายนอกแค่เรียก player.draw(g2, cameraX, cameraY, zoom) โดยไม่ต้องรู้ว่าใช้ sprite ไหน, ต้องตัดภาพกี่เฟรม

Enemy Class

```
protected abstract void onHurt();
protected abstract void onDie();

public abstract void update();
public abstract void draw(Graphics2D g2, int cameraX, int cameraY, double zoom);
```

ใช้ abstract เพื่อกำหนด พฤติกรรมที่ต้องมีแต่ยังไม่ระบุรายละเอียด ทำให้คลาสลูก (เช่น Boss) ต้อง implement เมธอดเหล่านี้เอง เช่น การตาย, การโจมตี, การวาดภาพ เป็นการบังคับให้โครงสร้างทุก Enemy มีรูปแบบเดียวกันแต่แตกต่างในรายละเอียด

Boss Class

```
@Override
public void update() {
    if (!alive && state != State.DEAD) {
        onDie();
    }

    commonInvincibleTick();

    atkCooldown = Math.max(0, atkCooldown - 1);

    switch (state) {
```

```
@Override
public void draw(Graphics2D g2, int cameraX, int cameraY, double zoom) {
    BufferedImage img = pickImage();

    int feetX = worldX + solidArea.x + solidArea.width / 2;
    int feetY = worldY + solidArea.y + solidArea.height;
```

คลาส Enemy เป็น abstract class ของ Boss ต้อง override เมธอด เช่น update() และ draw() ทำให้แต่ละศัตรูสามารถมีพฤติกรรมเฉพาะได้ โดยไม่ต้องเขียนโค้ดซ้ำใน Enemy

2.7.6 Polymorphism (พหุสัณฐาน)

GamePanel Class

```
player.update();
updateCamera();

warpEffect.update();

if (currentMapId == 1 && boss != null && boss.isAlive()) {

    boss.update();
```

ทั้ง Player และ Boss มีเมธอด update() ที่ชื่อเหมือนกัน แต่ทำงานต่างกัน (เช่น การเคลื่อนไหว, การโจมตี, การตรวจชน ฯลฯ) แสดงให้เห็นการใช้ Polymorphism ผ่าน Overriding

TileManager2 Class

```
public boolean isSolidAtPixel(int px, int py) {
    int tx = tileWidth  == 0 ? 0 : px / tileWidth;
    int ty = tileHeight == 0 ? 0 : py / tileHeight;
    if (tx < 0 || ty < 0 || tx >= mapWidth || ty >= mapHeight) return true;
    return collisionMap[ty][tx];
}

public boolean isTopTileAtPixel(int px, int py) {
    int tx = tileWidth  == 0 ? 0 : px / tileWidth;
    int ty = tileHeight == 0 ? 0 : py / tileHeight;
    if (tx < 0 || ty < 0 || tx >= mapWidth || ty >= mapHeight) return false;
    for (Tile2[][] layer : topLayers) if (layer[ty][tx] != null) return true;
    return false;
}
```

ทั้งสองเมธอดใช้การคำนวณพิกัดของ Tile จากพิกเซลเหมือนกัน แต่ตอบสนองต่างกัน (อันหนึ่งตรวจชน, อีกอันตรวจ top layer) ฟังก์ชัน drawTile() และ render() ก็ใช้แนวคิดนี้ในการวาดสิ่งต่างๆ ที่เป็น Tile โดยสามารถใช้กับ Tile หรือ Player ได้เหมือนกันผ่านการจัดเรียง RenderBucket

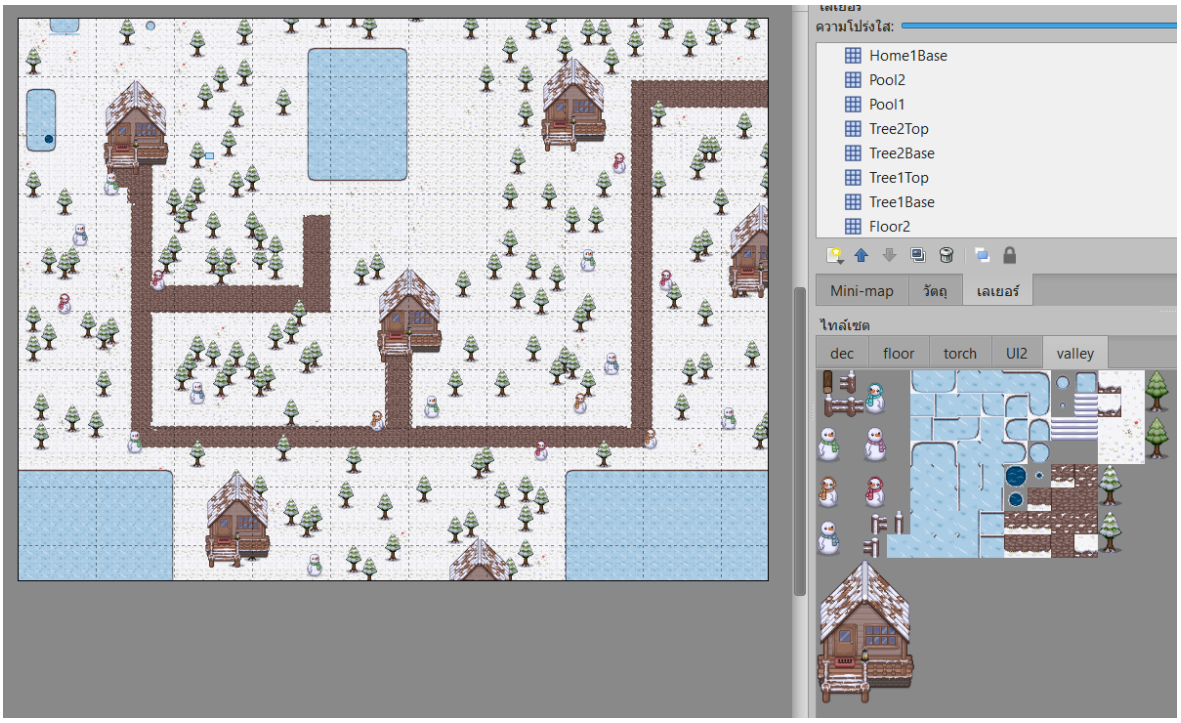
Player Class

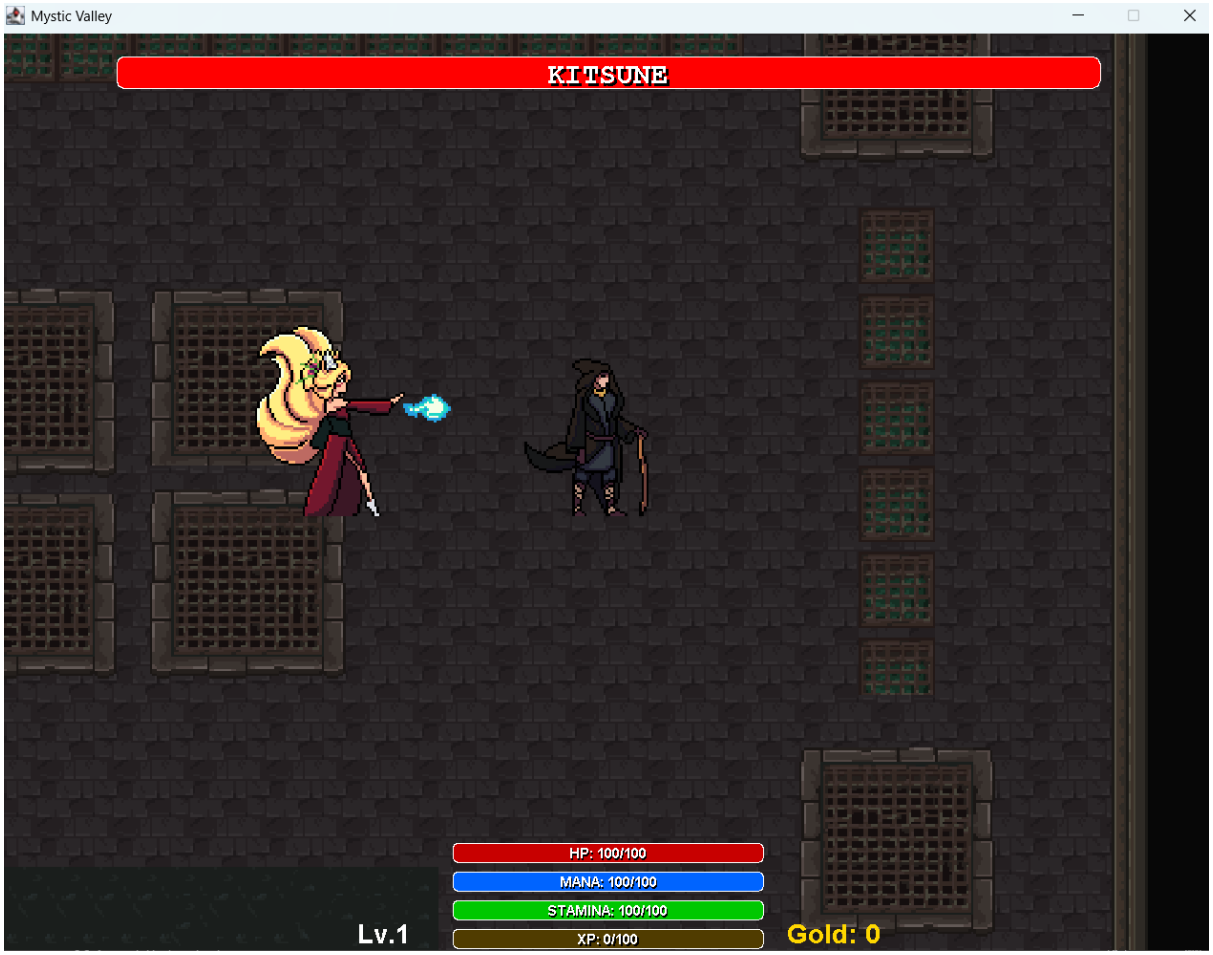
```
public Player(GamePanel gp, KeyHandler keyH) {
    this.gp = gp;
    this.keyH = keyH;
    setDefaultValues();
    getPlayerImage();
}
```

หน้าที่สร้างวัตถุ Player ขึ้นมา พร้อมกำหนดค่าเริ่มต้น เช่น ความเร็ว, ทิศทาง, และโหลดภาพ Sprite ของตัวละคร เป็นตัวอย่างของการกำหนดสภาวะเริ่มต้นของ Object เมื่อถูกสร้าง (Initialization)

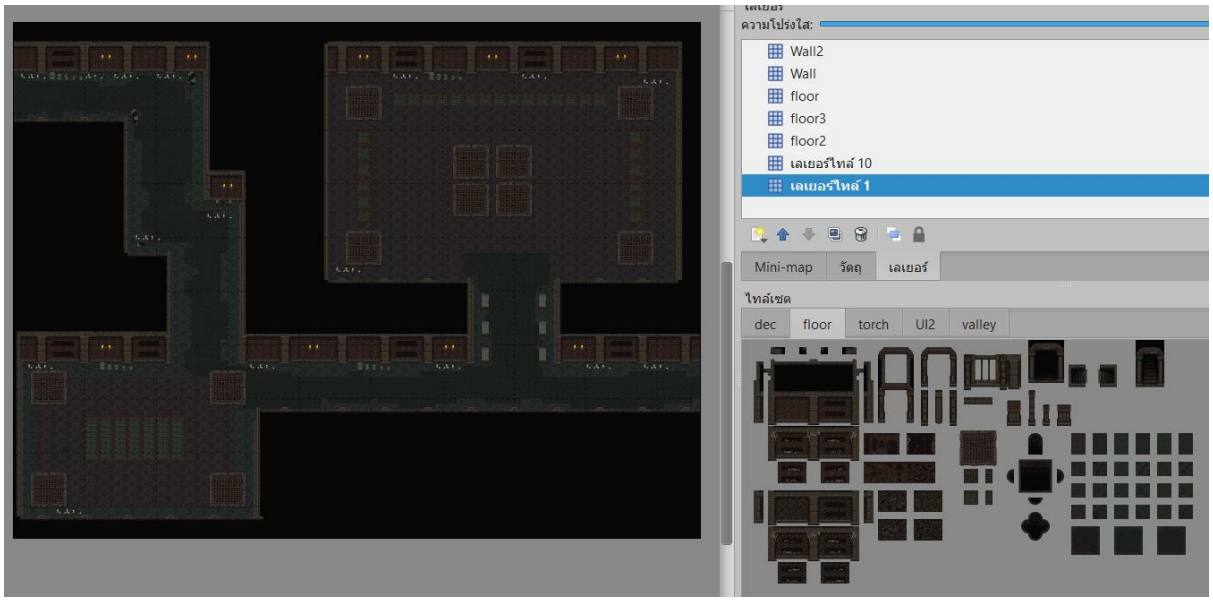
2.8 GUI

เมื่อเริ่มเกมเข้ามาผู้เล่นจะเกิดอยู่ที่แผนที่หมู่บ้านซึ่งผู้เล่นสามารถเดินสำรวจแมพได้ มีแถบ UI ด้านล่างเพื่อแสดงค่าสถานะต่าง ๆ





เมื่อผู้เล่นเดินเข้าจุดวาร์ปที่หมู่บ้าน ผู้เล่นจะวาร์ปเข้าสู่ดันเจี้ยนที่มีบอส จะมี UI แทบสถานะค่าเลือดของบอสอยู่



Components

JPanel: พื้นที่หลักของเกม (ใช้ paintComponent() วาดกราฟิก)

Graphics2D: ใช้วาดภาพ, ตัวละคร, UI bar

BufferedImage: โหลดภาพจากไฟล์ (พื้นหลัง, UI, Boss, Player)

Layout

ใช้ manual drawing layout (ไม่ใช้ Layout Manager) การจัดวางตำแหน่งทุกอย่างคำนวณเองใน paintComponent() เช่น barX, barY, cameraX, cameraY เพราะใช้พิกัดพิกเซลโดยตรง

2.9 Event Handling

2.9.1) **Mouse Events** ใช้เพื่อควบคุมผู้เล่นให้โจมตี

```
addMouseListener(new java.awt.event.MouseAdapter() {
    @Override
    public void mousePressed(java.awt.event.MouseEvent e) {
        if (!player.isAttacking && !player.isDead) {
            player.startAttack();
        }
    }
});
```

2.9.2) **Keyboard Events** ใช้เพื่อควบคุมผู้เล่นให้เดินบน, ซ้าย, ล่าง, ขวาผ่าน W, A, S, D และวิ่งเมื่อผ่าน SHIFT

```java
@Override
public void keyPressed(KeyEvent e) {

    int code = e.getKeyCode();

    if (code == KeyEvent.VK_W) {
        upPressed = true;
    }
    if (code == KeyEvent.VK_S) {
        downPressed = true;
    }
    if (code == KeyEvent.VK_A) {
        leftPressed = true;
    }
    if (code == KeyEvent.VK_D) {
        rightPressed = true;
    }
    if (code == KeyEvent.VK_SHIFT) {
        shiftPressed = true;
    }
    if (code == KeyEvent.VK_J) {
        gp.player.takeDamage(20);
    }
```

```java
@Override
public void keyReleased(KeyEvent e) {

    int code = e.getKeyCode();

    if (code == KeyEvent.VK_W) {
        upPressed = false;
    }
    if (code == KeyEvent.VK_S) {
        downPressed = false;
    }
    if (code == KeyEvent.VK_A) {
        leftPressed = false;
    }
    if (code == KeyEvent.VK_D) {
        rightPressed = false;
    }
    if (code == KeyEvent.VK_SHIFT) {
        shiftPressed = false;
    }
```




2.10 Algorithm

2.10.1) Fixed Time Step Game Loop

```java
@Override
public void run() {

    // Thread

    double drawInterval = 1000000000 / 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--;
        }
    }
}
```

ข้อดี เกมทำงานเร็วเท่ากันทุกเครื่อง (Frame Rate Independent) การเคลื่อนไหวคำนวณได้แม่นยำ

2.10.2) Camera System Algorithm

```java
public void updateCamera() {

    cameraX = (int) (player.worldX + player.solidArea.x + player.solidArea.width / 2
            - screenWidth / (2 * zoomFactor));
    cameraY = (int) (player.worldY + player.solidArea.y + player.solidArea.height / 2
            - screenHeight / (2 * zoomFactor));

    if (currentMap != null) {

        int mapW = currentMap.getMapWidth() * currentMap.getTileWidth();
        int mapH = currentMap.getMapHeight() * currentMap.getTileHeight();

        int viewW = (int) (screenWidth / zoomFactor);
        int viewH = (int) (screenHeight / zoomFactor);

        cameraX = Math.max(0, Math.min(cameraX, mapW - viewW));
        cameraY = Math.max(0, Math.min(cameraY, mapH - viewH));
    }
}
```

มีหลักการทำงาน คือ หาจุดกึ่งกลางของ Player คำนวณตำแหน่งกล้องให้ Player อยู่กึ่งกลางหน้าจอ คำนวณขนาด Map และ Viewport และ Clamp กล้องไม่ให้เกินขอบแผนที่

2.10.3) Collision Detection Algorithm

```
private boolean collidesAtOffset(int ox, int oy, TileManager2 tm) {
    int left = worldX + ox + solidArea.x;
    int top = worldY + oy + solidArea.y;
    int right = left + solidArea.width - 1;
    int bottom = top + solidArea.height - 1;

    return tm.isSolidAtPixel(left, top) || tm.isSolidAtPixel(right, top) || tm.isSolidAtPixel(left, bottom)
           || tm.isSolidAtPixel(right, bottom);
}
```

```
public boolean isSolidAtPixel(int px, int py) {
    int tx = tileWidth  == 0 ? 0 : px / tileWidth;
    int ty = tileHeight == 0 ? 0 : py / tileHeight;
    if (tx < 0 || ty < 0 || tx >= mapWidth || ty >= mapHeight) return true;
    return collisionMap[ty][tx];
}
```

มีหลักการทำงาน คือ คำนวณ 4 มุมของ Hitbox หลังเคลื่อนที่ ตรวจสอบทั้ง 4 มุม แปลง Pixel เป็น Tile Coordinate ตรวจสอบขอบเขต และเช็ค Collision Map

2.10.4) Boss AI Decision Tree And Dynamic Difficulty

```
private void decideAndAct() {
    int d = distToPlayer();
    if (d > vision) {
        // idle
        state = State.IDLE;
        animate( idleR.length );
        return;
    }

    // face the player
    facing = dirToPlayer();

    if (d <= meleeRange && atkCooldown == 0) {
        state = State.ATK1;
        frameCounter = frameIndex = 0;
        atkCooldown = atkInterval;
        return;
    }

    if (d <= 220 && atkCooldown == 0) {
        state = State.ATK3; // forward shots
        frameCounter = frameIndex = 0;
        atkCooldown = atkInterval;
        return;
    }

    // occasionally do radial burst
    if (atkCooldown == 0) {
        state = State.ATK2;
        frameCounter = frameIndex = 0;
        atkCooldown = atkInterval + 60;
        return;
    }

    // chase
    state = State.WALK;
    chasePlayer();
    animate( walkR.length );
}
```

```
private int distToPlayer() {
    int cx = worldX + solidArea.x + solidArea.width/2;
    int cy = worldY + solidArea.y + solidArea.height;
    int px = gp.player.worldX + gp.player.solidArea.x + gp.player.solidArea.width/2;
    int py = gp.player.worldY + gp.player.solidArea.y + gp.player.solidArea.height;
    int dx = px - cx;
    int dy = py - cy;
    return (int)Math.hypot(dx, dy);
}

private String dirToPlayer() {
    int cx = worldX + solidArea.x + solidArea.width/2;
    int px = gp.player.worldX + gp.player.solidArea.x + gp.player.solidArea.width/2;
    return (px >= cx) ? "right" : "left";
}
```

หลักการ คือ คำนวณระยะทาง เงื่อนไข 1: อยู่นอก Vision, เงื่อนไข 2: อยู่ในระยะ Melee (Priority สูงสุด), เงื่อนไข 3: อยู่ในระยะกลาง, เงื่อนไข 4: อยู่ไกล, เงื่อนไข 5: อยู่ใน Cooldown

```
public void scaleWithPlayer(int playerLevel) {

    this.maxHP = BASE_HP + (playerLevel * 50);
    this.HP = this.maxHP;

    this.damageBoost = playerLevel * 2;

    this.meleeRange = 60 + (playerLevel * 3);
    this.vision = 360 + (playerLevel * 2);

}
```

2.10.5) Depth Sorting Algorithm (Y-Sorting)

```
public void render(Graphics2D g, int camX, int camY, int sw, int sh, double zoom, Player player) {

    // Base
    for (Tile2[][] layer : baseLayers) {
        for (int y=0; y<mapHeight; y++)
            for (int x=0; x<mapWidth; x++)
                if (layer[y][x]!=null) drawTile(g, layer[y][x], x, y, camX, camY, sw, sh, zoom);
    }

    // Buckets + player depth
    List<RenderBucket> bucket = new ArrayList<>();

    for (Tile2[][] layer : topLayers) {
        for (int y=0; y<mapHeight; y++) {
            for (int x=0; x<mapWidth; x++) {
                Tile2 t = layer[y][x]; if (t==null) continue;
                final int tx=x, ty=y; final Tile2 tt=t;
                bucket.add(new RenderBucket(){{
                    sortY = tt.worldBottomY;
                    drawCall = () -> drawTile(g, tt, tx, ty, camX, camY, sw, sh, zoom);
                }});
            }
        }
    }

    bucket.add(new RenderBucket(){{
        sortY = player.depthY;
```

หลักการ คือ วาด Base Layers สร้าง Bucket สำหรับ Depth Sorting เพิ่ม Top Layer Tiles เพิ่ม Player เรียงลำดับ (เล็ก ⟶ ใหญ่) วาดตามลำดับ

2.10.6) Boss Attack Hit Detection

```
private void checkMeleeHit() {

    Rectangle bossHit = new Rectangle(
        worldX + solidArea.x,
        worldY + solidArea.y,
        solidArea.width,
        solidArea.height
    );

    Rectangle playerHit = new Rectangle(
        gp.player.worldX + gp.player.solidArea.x,
        gp.player.worldY + gp.player.solidArea.y,
        gp.player.solidArea.width,
        gp.player.solidArea.height
    );

    if (bossHit.intersects(playerHit)) {

        String dir = (facing.equals("left") ? "left" : "right");

        int dmg = 5 + damageBoost;


        // Object Interaction
        gp.player.takeDamageFrom(dir, dmg, true, false, 0);
    }
}
```

หลักการ คือ ตรวจสอบการชนกันของ Boss Hitbox และ Player Hitbox

2.10.7) Knockback Physics

```
public void takeDamageFrom(String dir

    if (isDead || invincible)
        return;

    HP -= dmg;
    isHurt = true;
    hurtFrame = 1;
    hurtCounter = 0;

    if (dir != null)
        facingDirection = dir;

    if (knockback) {
        final int kbX;
        final int kbY;

        switch (dir) {
        case "left":
            kbX = 5;
            kbY = 0;
            break;
        case "right":
            kbX = -5;
            kbY = 0;
            break;
        case "up":
            kbX = 0;
            kbY = 5;
            break;
        case "down":
            kbX = 0;
            kbY = -5;
            break;
        default:
            kbX = 0;
            kbY = 0;
            break;
        }
```

```
        new Thread(() -> {
            for (int i = 0; i < 6; i++) {
                moveWithCollision(kbX, kbY);
                try {
                    Thread.sleep(10);
                } catch (Exception ignored) {
                }
            }
        }).start();
    }
```

หลักการ คือ กำหนดทิศทางและแรงและใช้ Thread ทำ Knockback Animation

2.10.8) Level Up Calculation

```
public void gainXP(int amount) {
    XP += amount;

    // Level Up!
    while (XP >= maxXP) {
        levelUp();
        if (gp.boss != null) {
            gp.boss.onHurt();
        }
    }
}

private void levelUp() {
    level++;
    XP -= maxXP;
    maxXP += 20;

    // Reset level (30)
    if (level >= 30) {
        level = 0;
        XP = 0;
        maxXP = 100;
    }

    // STATUS UP BY LEVEL
    if (level > 0) {
        maxHP += 15;
        maxMana += 10;
        maxStamina += 10;

        // RE
        HP = maxHP;
        mana = maxMana;
        stamina = maxStamina;

        // BOSS UP BY LAVEL
        if (gp.boss != null) {
            gp.boss.scaleWithPlayer(level);
        }
    }
}
```

2.10.9) Stamina System Algorithm

```
// RUN STAMINA SYSTEM
private void updateStamina() {
    if (isRunning) {
        // USE STAMINA RUN
        stamina -= 1;
        if (stamina <= 0) {
            stamina = 0;
            staminaDepleted = true;
            isRunning = false;
        }
    } else {
        // RE-STAMINA (NOT RUN)
        if (stamina < maxStamina) {
            stamina += 1;
        }

        //
        if (stamina >= maxStamina * 0.1) {
            staminaDepleted = false;
        }
    }
}
```

2.10.10) Warp Fade Transition

```
public WarpEffect(GamePanel gp) {
    this.gp = gp;
}

public boolean teleporting = false;

public void triggerWarp(int targetMapID) {
}

public void update() {

    float speed = 0.02f;

    if (fadingOut) {
        alpha += speed;
        if (alpha >= 1f) {
            alpha = 1f;
            fadingOut = false;
            gp.doWarp(nextMapID);

            fadingIn = true;
        }
    } else if (fadingIn) {
        alpha -= speed;
        if (alpha <= 0f) {
            alpha = 0f;
            fadingIn = false;
        }
    }

    if (!fadingOut && !fadingIn) {
        teleporting = false;
    }
}

public void draw(Graphics2D g, int screenW, int screenH) {
    if (!fadingOut && !fadingIn) return;

    g.setComposite(java.awt.AlphaComposite.getInstance(
            java.awt.AlphaComposite.SRC_OVER, alpha));
```

2.10.11) Attack Animation & Hit Detection

```
if (isAttacking) {

    attackCounter++;

    if (attackFrame == 4) {
        checkAttackHit();
    }

    if (attackCounter > 10) {
        attackFrame++;
        attackCounter = 0;

        if (attackFrame > 7) {
            attackFrame = 1;
            isAttacking = false;
        }
    }
    return;
}
```

2.10.12) Boss Attack Frame Timing

```
private void updateAttack() {
    frameCounter++;
    int max = switch (state) {
        case ATK1 -> atk1R.length;
        case ATK2 -> atk2R.length;
        case ATK3 -> atk3R.length;
        default -> 1;
    };

    if (frameCounter > 6) { // animation speed
        frameCounter = 0;
        frameIndex++;

        // Melee attack hit frame
        if (state == State.ATK1 && (frameIndex == 3 || frameIndex == 4)) {
            checkMeleeHit();
        }

        // Projectile spawn
        if (state == State.ATK2 && frameIndex == 4)
            spawnRadial();

        if (state == State.ATK3 && (frameIndex == 3 || frameIndex == 5))
            spawnForward();
    }

    if (frameIndex >= max) {
        frameIndex = 0;
        state = State.IDLE;
    }
}
```

2.10.13) Tile Flip Transformation

```
private BufferedImage applyFlip(BufferedImage src, boolean flipH, boolean flipV, boolean flipD) {
    if (!flipH && !flipV && !flipD) return src;
    BufferedImage out = new BufferedImage(tileWidth, tileHeight, BufferedImage.TYPE_INT_ARGB);
    Graphics2D g2 = out.createGraphics();
    AffineTransform at = new AffineTransform();
    if (flipD) { at.translate(tileHeight, 0); at.rotate(Math.toRadians(-90)); }
    if (flipH) { at.translate(tileWidth, 0); at.scale(-1, 1); }
    if (flipV) { at.translate(0, tileHeight); at.scale(1, -1); }
    g2.drawImage(src, at, null); g2.dispose();
    return out;
}
```

บทที่ 3
สรุปผลและข้อเสนอแนะ

3.1 ในการพัฒนาเกมพบปัญหาหลัก ๆ ดังนี้

- โค้ดกระจายอยู่ในหลายคลาส ซึ่งบางจุดมีการเรียกข้ามกันโดยยังไม่มี interface หรือระบบกลางรองรับ ปัญหาที่พบบ่อยคือ เมธอดหรือฟิลด์ไม่สามารถเรียกได้

- ระบบแมพและการเรนเดอร์ Tile ปัญหาการ “บังหัว” หรือ “หลังคาหาย” มาจากลำดับการวาด Layer

- เครื่องมือและ Workflow การพัฒนา ใช้ Eclipse + Java Swing ซึ่งเหมาะกับการฝึก OOP แต่ค่อนข้างจำกัดสำหรับเกม 2D ไม่มีระบบ asset manager รวมกลาง ทำให้เวลาเปลี่ยน path ต้องแก้หลายที่

3.2 จุดเด่นของโปรแกรม

- ระบบกล้อง (Dynamic Camera System) กล้องในเกมสามารถ “ติดตามผู้เล่นแบบนุ่มนวล” โดยคำนวณจากตำแหน่งจริงของตัวละครและขนาดหน้าจอ รองรับระบบ Zoom In / Out

- ระบบการต่อสู้ (Combat & Boss System) มีระบบ Boss ที่มีหลายเฟส เช่น การเดิน, โจมตี, ตาย และเกิดใหม่ ระบบนี้รวมการทำงานของ Animation, Collision, HP bar และพฤติกรรม AI

- ระบบวาร์ปและการเปลี่ยนแผนที่ (Warp & Map Transition System) ผู้เล่นสามารถ “วาร์ประหว่างหมู่บ้านและดันเจี้ยน”

- ระบบการเรนเดอร์แบบหลายเลเยอร์ (Layered Map Rendering) รองรับ “การซ่อน-แสดง” หลังคาเมื่อตัวละครอยู่ด้านล่าง

3.3 ข้อเสนอแนะ

- ควรเพิ่มระบบบันทึก

- ปรับปรุงการตอบสนองของการชน ให้มีความแม่นยำมากขึ้น เพื่อให้การเล่นลื่นไหล

- ปรับปรุงระบบศัตรูและบอส ระบบ Boss ปัจจุบันมีพฤติกรรมพื้นฐานแล้ว แต่ยังไม่มีระบบ AI หรือการเปลี่ยนเฟส (Phase) ใช้ Finite State Machine (FSM) สำหรับควบคุมพฤติกรรมของ Boss เช่น Idle ⟶ Chase ⟶ Attack ⟶ Dead เพิ่มระบบ Drop Item / EXP / Level Scaling

- เสริมระบบภาพและเสียงเพื่อเพิ่มอรรถรส