โครงงาน

ยาวๆ ขาวๆ 2.5D
(Long white thingy 2.5D)

จัดทำโดย

6504062620175 นายอธิชา เล็กสรรเสริญ

เสนอ

ผู้ช่วยศาสตราจารย์สถิตย์ ประสมพันธ์

วิชา 040613204 Object-Oriented Programming

ภาคเรียนที่ 1 ปีการศึกษา 2566

ภาควิชาวิทยาการคอมพิวเตอร์และสารสนเทศ คณะวิทยาศาสตร์ประยุกต์
มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ

เกี่ยวกับโครงงาน

ชื่อโปรเจค: ยาวๆ ขาวๆ 2.5D (Long white thingy 2.5D)
นำเสนอโดย: นายอธิชา เล็กสรรเสริญ
อาจารย์ผู้สอน: ผู้ช่วยศาสตราจารย์สถิตย์ ประสมพันธ์
Source Code: https://github.com/tonkaew131/JavaGameProject

บทที่ 1 ที่มาและความสำคัญของโครงการ

โครงงานนี้จัดขึ้นเพื่อวัดผลความสามารถในการเรียนวิชา Object Oriented Programming โดยการนำเรื่องที่เรียนมาสร้างเป็นชิ้นงานในรูปแบบของเกม โดยใช้แนวคิดการเขียนโปรแกรมแบบเชิงวัตถุ และยังช่วยให้ผู้จัดทำเรียนรู้อุปกรณ์และเครื่องมือ ผู้จัดทำได้สร้างเกมนี้ขึ้นมา

ประเภทของโครงการ

เกม (Game)

ประโยชน์

1. ฝึกวิธีการแก้ปัญหา
2. ฝึกไหวพริบในการเอาตัวรอด
3. บรรเทาความเครียด
4. เพื่อแนวคิด การเขียนโปรแกรมแบบเชิงวัตถุมาประยุกต์ใช้

ขอบเขตของโครงการ

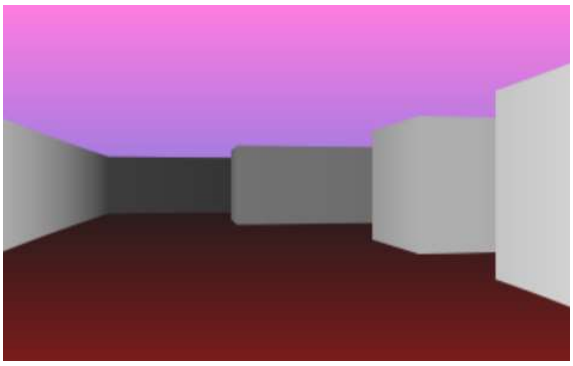
1. ตารางเวลาการดำเนินโครงการ (Project Schedule)

งาน	สัปดาห์ 1 (ก.ย.)			สัปดาห์ 2 (ต.ค.)			สัปดาห์ 3 (ต.ค.)			สัปดาห์ 4 (ต.ค.)			สัปดาห์ 5 (ต.ค.)		
	25	**27**	**29**	**2**	**4**	**6**	**9**	**11**	**13**	**16**	**18**	**20**	**23**	**25**	**27**
1. Renderer / Ray cast															
2. Player Control															
3. Collectibles															
4. Ghost AI															
5. Soung system															
	0%	6%	13%	22%	28%	34%	44%	50%	56%	66%	72%	78%	88%	94%	100%

บทที่ 2 **การพัฒนา**

เนื้อเรื่องย่อ

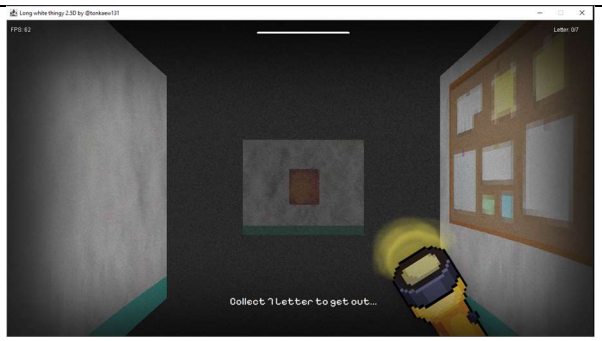
เมื่อคืนนึงคุณตื่นขึ้นมาในบ้านร้างพร้อมกับไฟฉาย แต่ดันมีเสียงลึกลับจากที่ไหนไม่รู้ที่คุณไม่รู้ มาบอกให้คุณเก็บจดหมาย 7 อันเพื่อจะพาคุณออกจากที่นี้ โดยเนื่องจากบ้านหลังผ่านคดีโชกโชนมามากมาย จึงมีผีมาตามค่อยให้กำลังใจคุณ โดยเกมจะเป็นรูปแบบมุมมองบุคคลที่หนึ่งจากการ Ray Casting แมพ 2 มิติให้ออกมาเป็นภาพลวงตา 3มิติ คล้ายกับเกม Wolfenstein 3D



ตัวอย่าง ภาพลวงตา 3มิติ

เนื้อเรื่องย่อ

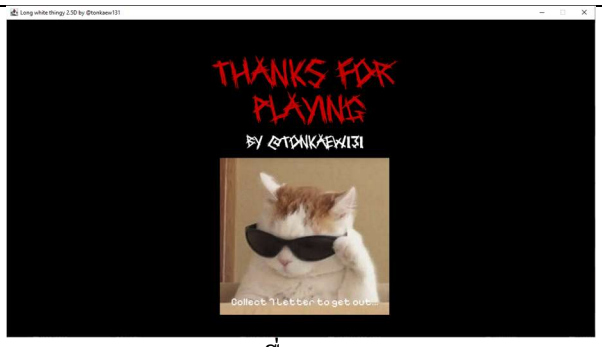
ใช้ W, A, S, D ในการเดินและหมุนมุมกล้อง บนพื้นที่จำลอง 3มิติ และเมื่อเจอจดหมายสามารถกดเก็บด้วย E ได้ โดยจะมี จำนวนจดหมายที่เก็บแล้วขึ้นบนหน้าจอ หาจดหมาย 7 อันให้เร็วที่สุด



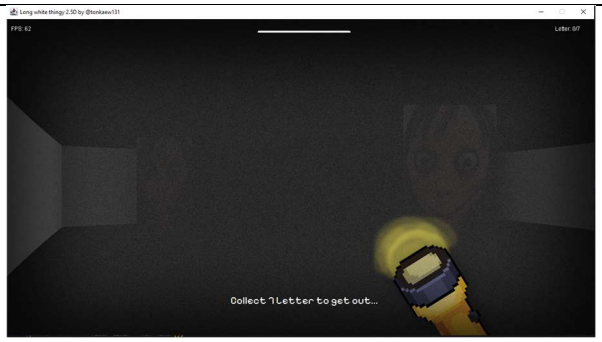
บรรยากาศเกม

จดหมาย

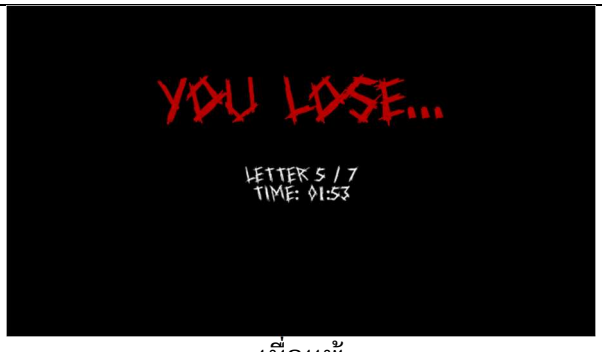
เมื่อยังเก็บจำหมายไม่ครบ



เมื่อชนะ



การแสดง Sprite



เมื่อแพ้

แผนภาพ Class Diagram

โครงการนี้จะมีคราสหลัก 10 คราสคือ

1. คราส Player จะเก็บข้อมูลของผู้เล่น
2. คราส Map จะเก็บข้อมูลของแผนที่ในเกม ละจุดจดหมายต่างๆ
3. คราส RayCast จะเป็นคราสคำนวณการ Ray casting
4. คราส Sprite จะเก็บข้อมูลรายละเอียดสิ่งมีชีวิตในเกม
5. คราส Renderer เป็นคราสหลักในการแสดงผลทุกอย่าง

6. คราส Tick เป็นคราสที่รัน Thread เพื่อรัน Logic เบื้องหลังต่างๆ ของเกม
7. คราส Texture เป็นคราสที่โหลดไฟล์ Texture
8. คราส KeyListener เป็นคราสที่รับ Event Keyboard จาก User
9. คราส Game เป็นคราสหลักที่เรียกใช้วัตถุต่างๆ
10. คราส Sound เป็นคราสระบบเสียงในเกม

รูปแบบการพัฒนาโครงการ

- ภาษา: Java
- GUI: javax.swing
- ระบบเสียง: javax.sound.sampled
- โปรแกรมวาดรูป / ตัดรูป: Photoshop, Krita
- โปรแกรมวาดแผนที่: Excel

แนวคิดการเขียนโปรแกรมเชิงวัตถุ

- **Constructor**

```
public Renderer() {
    rayCaster = new RayCast();

    super.setSize(Setting.WINDOWS_WIDTH, Setting.WINDOWS_HEIGHT);
    super.setPreferredSize(new Dimension(Setting.WINDOWS_WIDTH, Setting.WINDOWS_HEIGHT));

    this.setLayout(new BorderLayout());

    titleLabel.setFont(FontCustom.PixelifySans.deriveFont(Font.BOLD, 64));
    titleLabel.setForeground(Color.WHITE);
    titleLabel.setBorder(new EmptyBorder(100, 0, 0, 0)); // top, left, bottom, right
    add(titleLabel, BorderLayout.CENTER);

    subtitleLabel.setFont(FontCustom.PixelifySans.deriveFont(Font.PLAIN, 22));
    subtitleLabel.setForeground(Color.WHITE);
    subtitleLabel.setBorder(new EmptyBorder(0, 0, 100, 0)); // top, left, bottom, right
    add(subtitleLabel, BorderLayout.SOUTH);

    zBuffer = new double[Setting.WINDOWS_WIDTH];

    this.setBackground(Color.BLACK);
    timer.start();
}
```

Constructor ของคราส Renderer จะมีการตั้งขนาดของจอ และเพิ่ม Components ต่างๆ และสร้าง Array zBuffer และเริ่ม Timer

```
public Map() {
    addLetter(5, 26);
    addLetter(9, 22);
    addLetter(19, 0);
    addLetter(25, 7);
    addLetter(37, 24);
    addLetter(32, 16);
    addLetter(18, 24);

    parseMap();
}
```

```
public RayCast() {
}

public RayCast(Point<Double> playerPosition, double direction) {
    this.playerPosition = playerPosition;
    this.direction = direction;
}
```

Constructor ของคราส Map จะเพิ่มตำแหน่งของจดหมายต่างๆ และทำการแปลงข้อมูลของ Texture Id ไปเป็น Enum ของคราส Texture

Constructor ของคราส RayCast จะมีทั้งแบบ Default และแบบมี Perimeter โดย Perimeter จะนำมาเซ็ท Attribute

```
public Sound() {
    if (Setting.ENABLED_SOUND) {
        Clip backgroundMusic = loadSound("/resources/sound/background_music.wav");
        backgroundMusic.start();
        backgroundMusic.loop(Clip.LOOP_CONTINUOUSLY);
    }
}
```

Constructor ของคราส Sound จะทำงานโหลดเสียงเพลงพื้นหลังและเริ่มเล่น โดยเพลงพื้นหลังจะเริ่มเล่นใหม่ไปเรื่อยๆ เมื่อจบ

```
// src\core\Sprite.java
public Sprite(Point<Double> pos, double width, double height, String imagePath) {
    this.pos = pos;
    this.width = width;
    this.height = height;

    try {
        this.image = ImageIO.read(getClass().getResourceAsStream(imagePath));
        this.imageHeight = image.getHeight(null);
        this.imageWidth = image.getWidth(null);
    } catch (IOException e) {
        System.out.println(String.format("[Sprite]: Failed to load sprite texture! (%s)", imagePath));
        e.printStackTrace();
        return;
    }
}

// src\core\SpriteWhite.java
public SpriteWhite(Point<Double> pos) {
    super(pos, 1.0, 1.0, "/resources/texture/momo_ghost.png");
}
```

Constructor ของคราส Sprite จะรับ Perimeters ต่างๆ เข้ามาและเซ็ท Attribute รวมถึงการอ่านไฟล์รูปของ Sprite นั้นๆ ส่วนคราส SpriteWhite จะเรียก Constructor ของคราสแม่ (Sprite) มาแต่จะรับ Perimeter แค่เพียงตำแหน่งของ SpriteWhite

```
private Texture(int textureId) {
    this.textureId = textureId;
}
```

Constructor ของคราส Texture จะเป็น Private constructor เนื่องจากเป็น Enum จำเป็นจะต้องเซ็ทค่าภายในเป็น Id ที่เป็นตัวเลข

```
public Tick(Renderer renderer, Player player) {
    this.player = player;

    this.renderer = renderer;
    this.renderer.setTick(this);

    long currentTimeMillis = getCurrentMillis();
    gameStartMillis = currentTimeMillis;
    lastTickMillis = currentTimeMillis;
}
```

Constructor ของคราส Tick จะรับ Perimeter เข้ามาเซ็ทค่า Attribute ต่างๆ และจะตั้งค่าของ gameStartMillis (เก็บเวลาที่เกมเริ่ม) เป็นเวลาปัจจุบัน

- **Encapsulation & Composition**

```
// src\core\Renderer.java
private RayCast rayCaster;
private Map map;
private Tick tick;
private Timer timer = new Timer((int) (1000.0 / Setting.MAX_FPS), this);
private Player player;
```

Attribute ของเกือบทุกคราสจะเป็น Private และจะมี methods Getter และ Setter ในการเปลี่ยนแปลงค่าเพื่อเช็คความถูกต้องของข้อมูล

ในคราสต่างๆ จะมีการ Composite วัตถุมาจากคราสอื่นเพื่อดึงข้อมาเช็คหรือแก้ไขข้อมูลเช่น method การเก็บจดหมายของคราส Player ก็จะเรียกใช้วัตถุจากคราส RayCast และ Map เพื่อเช็คว่าผู้เล่นสามารถเก็บจดหมายได้หรือไม่

```
// src\core\Player.java
public void collect() {
    rayCast.setDirection(directionAlpha);
    rayCast.setPlayerPosition(getPosition());
    rayCast.cast();

    if (rayCast.getDistance() > Setting.LETTER_REACH_DISTANCE)
        return;

    Point<Integer> mapCheck = rayCast.getMapPoint();
    if (map.checkLetter(mapCheck.x, mapCheck.y)) {
        map.removeLetter(mapCheck.x, mapCheck.y);
        letterCount++;

        sound.playLetterPickupSound();
    }
    return;
}
```

- **Polymorphism & Inheritance**

```java
// src\core\SpriteWhite.java
public class SpriteWhite extends Sprite {
    public SpriteWhite(Point<Double> pos) {
        super(pos, 1.0, 1.0, "/resources/texture/momo_ghost.png");
    }
}

// src\core\MapTest.java
private ArrayList<Sprite> renderedSprites = new ArrayList<>();

renderedSprites.add(new SpriteWhite(new Point<Double>(3d, 3d)));
renderedSprites.add(new SpriteWhite(new Point<Double>(1d, 1d)));
renderedSprites.add(new SpriteWhite(new Point<Double>(5d, 5d)));
renderedSprites.add(new SpriteWhite(new Point<Double>(2d, 5d)));
```

Sprite หรือสิ่งมีชีวิตต่างๆ (หรือผี) ในเกมจะมีคราสแม่คือคราส Sprite และจะสามารถ Inherit ออกเป็นหลากหลายรูปได้ แต่ทุก Sprite จะมีการ render ออกมาเหมือนกันจึงสามารถให้ SpriteWhite อยู่ในรูปของ Sprite ได้เลย

- **Abstract**

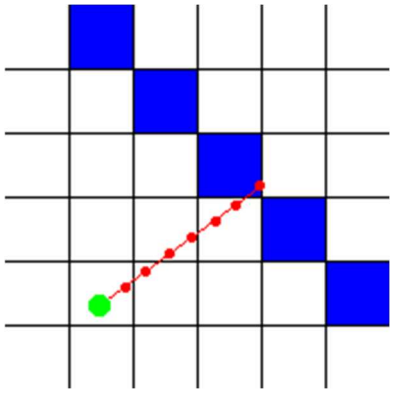
ขณะนี้ยังไม่มีการใช้ concept การ Abstract แต่จะสามารถมาประยุกต์ใช้กับคราส Map ในอนาคตได้ เช่นการมี abstract class Map โดยจะประกอบไปด้วย concrete method เช่นการ parseMap, checkLetter, ... แต่จะมี abstract method อย่างเช่นการ implements จุดต่างๆ ของแผนที่ที่สามารถ Interact ได้ เช่นการ spawn ผี หรือการตั้งตำแหน่งของจดหมาย

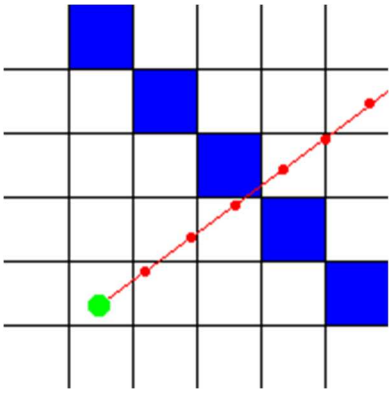
หรือจะเป็นการใช้กับคราส Sprite เพื่อสร้าง abstract method พฤติกรรมของผีตัวนั้นๆ ซึ่งจะสามารถนำไป implements เพื่อให้ผีแต่ละตัวมีพฤติกรรมที่แตกต่างกันได้

อัลกอริทึมที่สำคัญ

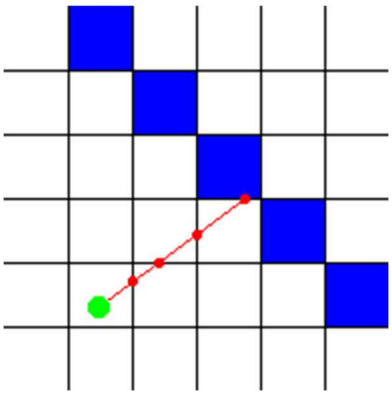
```
public void cast() {
    // DDA Algorithm
    double posX = playerPosition.x;
    double posY = playerPosition.y;

    Point<Double> endPoint = new Point<>(
            posX + Math.cos(direction) * 100,
            posY + Math.sin(direction) * 100);
    Point<Double> startPoint = new Point<>(posX, posY);

    Point<Double> rayDirection = new Point<>(0d, 0d);
    rayDirection.x = endPoint.x - startPoint.x;
    rayDirection.y = endPoint.y - startPoint.y;

    // Normalize
    double rayLength = Math.sqrt(rayDirection.x * rayDirection.x + rayDirection.y * rayDirection.y);
    rayDirection.x /= rayLength;
    rayDirection.y /= rayLength;

    // Point<Double> rayUnitStepSize = new Point<>(
    // Math.sqrt(1 + ((rayDirection.y / rayDirection.x) * (rayDirection.y /
    // rayDirection.x))),
    // Math.sqrt(1 + ((rayDirection.x / rayDirection.y) * (rayDirection.x /
    // rayDirection.y))));
    Point<Double> rayUnitStepSize = new Point<>(Math.abs(1.0d / rayDirection.x), Math.abs(1.0d / rayDirection.y));

    Point<Integer> mapCheck = new Point<>((int) posX, (int) posY);
    // store length in x, y
    Point<Double> rayLengthCumu = new Point<>(0d, 0d);
    // store step in x, y
    Point<Integer> rayStep = new Point<>(0, 0);

    if (rayDirection.x < 0) {
        rayStep.x = -1;
        rayLengthCumu.x = (startPoint.x - (double) mapCheck.x) * rayUnitStepSize.x;
    } else {
        rayStep.x = 1;
        rayLengthCumu.x = ((double) (mapCheck.x + 1) - startPoint.x) * rayUnitStepSize.x;
    }

    if (rayDirection.y < 0) {
        rayStep.y = -1;
        rayLengthCumu.y = (startPoint.y - (double) mapCheck.y) * rayUnitStepSize.y;
    } else {
        rayStep.y = 1;
        rayLengthCumu.y = ((double) (mapCheck.y + 1) - startPoint.y) * rayUnitStepSize.y;
    }

    Point<Double> beforeHit = new Point<>();

    boolean hit = false;
    double distance = 0;
    double MAX_DISTANCE = 100;
    while (!hit && distance < MAX_DISTANCE) {
        beforeHit.x = rayLengthCumu.x;
        beforeHit.y = rayLengthCumu.y;

        if (rayLengthCumu.x < rayLengthCumu.y) {
            mapCheck.x += rayStep.x;
            distance = rayLengthCumu.x;
            rayLengthCumu.x += rayUnitStepSize.x;
        } else {
            mapCheck.y += rayStep.y;
            distance = rayLengthCumu.y;
            rayLengthCumu.y += rayUnitStepSize.y;
        }

        if (mapCheck.x >= 0 && mapCheck.y >= 0 &&
                mapCheck.x < map.getMapWidth() && mapCheck.y < map.getMapHeight()) {
            if (map.getTexture(mapCheck.x, mapCheck.y) != Texture.EMPTY) {
                hit = true;
            }
        }
    }

    Point<Double> intersecPoint = new Point<>(0d, 0d);
    intersecPoint.x = startPoint.x + rayDirection.x * distance;
    intersecPoint.y = startPoint.y + rayDirection.y * distance;

    this.mapPoint = mapCheck;
    this.distance = distance;
    this.hitPoint = intersecPoint;
    this.hitSide = beforeHit.x < beforeHit.y ? HitSide.HORIZONTAL : HitSide.VERTICAL;
    return;
}
```

อัลกอริทึมหา DDA หรือ Digital Differential Analyzer เป็นอัลกอริทึมที่ใช้เพื่อการ Ray Casting เพื่อหาว่ามีกำแพงอยู่ที่ไหนและระยะห่างเท่าใด และสามารถหาตำแหน่งที่จุดนั้นตกบน Texture ได้ด้วย โดยที่ DDA เป็นอัลกอริทึมหลักที่ใช้ในการสร้างภาพ 3มิติเสมือนขึ้นมาอย่างมีประสิทธิภาพและรวดเร็ว เนื่องจากมีการคำนวณที่ประหยัด

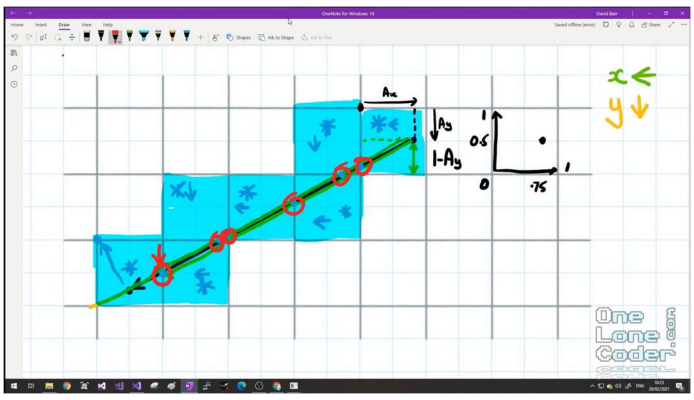




หากใช้การเช็คกำแพงแบบปกติ โดยการลูปไปเรื่อยๆ ทีละ step จนกว่าจะเจอกำแพง บ้างครั้งอาจทำให้เลยกำแพงได้ หรือถ้าหากใช้ step ที่เล็กลงจะทำให้ใช้ทรัพยากรในการคำนวณที่สูงมากๆ



แต่อัลกอริทึม DDA จะเช็คเพียงเส้นตัดของแกน x และแกน y เพียงเท่านั้น เนื่องจากเรารู้ว่าจากตำแหน่งปัจจุบันไปจุดตัดแกน x จะเพิ่ม y ทีละ 1 เสมอ และจากตำแหน่งปัจจุบันไปจุดตัดแกน y จะเพิ่ม x ที่ละ 1 เสมอ



บทที่ 3 **สรุป**

ปัญหาที่พบระหว่างการพัฒนา

1. เนื่องจากเวลาที่น้อยเกินไป ทำให้จำเป็นต้องลด Features ลงเยอะ
2. เนื่องจากเป็นแนวเกมที่ไม่ค่อยดัง จึงทำให้เนื้อหาเรียนรู้ หายาก และมักจะไม่ตรงกับที่ใช้
3. การคำนวณต่างๆ มีปัญหา ไม่สามารถใช้งานได้ทุกต้องในทุกๆ quadrant ทำให้จำเป็นต้องตัดหลายๆ Feature ออก
4. การคำนวณที่ใช้ CPU หนัก เวลาให้เขียนบนคอมที่ช้า จะปวดหัวเนื่องจาก Frame rate ที่น้อย

จุดเด่นของโปรแกรม

1. เป็นเกมสามมิติมุมมองบุคคลที่หนึ่ง ที่ไม่ได้ใช้ Game Engine หรือ Library ช่วยคำนวณ

2. มีระบบเสียงที่น่ากลัว

คำแนะนำสำหรับผู้สอนที่อยากให้อธิบาย หรือที่เรียนแล้วไม่เข้าใจ หรืออยากให้เพิ่มสำหรับน้องๆ รุ่นต่อไป

อยากให้อาจารย์ มาเฉลยแลปต่างๆ 😭 หลังจากหมดเวลาส่งแล้ว เนื่องจากบางข้อที่เป็นอัลกอริทึมที่ยากๆ จำเป็นต้องใช้ BFS, DFS หรือ Data structure รูปแบบต่างๆ มีบางคนอาจจะไม่เข้าใจ หรือ บางคนที่เข้าใจแล้ว อาจจะใช้วิธีที่ ไม่ได้ดีที่สุด 😕