โครงงาน

Mini project

จัดทำโดย

6504062630278 นายวิศรุต ศรีนวลปาน

เสนอ

ผู้ช่วยศาสตราจารย์สถิต ประสมพันธ์

วิชา Object Oriented Programming

ภาคเรียนที่ 1/2565

ภาควิชาวิทยาการคอมพิวเตอร์และสารสนเทศ คณะวิทยาศาสตร์ประยุกต์

มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ

เกี่ยวกับโครงงาน

ชื่อโปรเจค : Sorcerer's Odyssey

นำเสนอโดย : นายวิศรุต ศรีนวลปาน

อาจารย์ผู้สอน : ผู้ช่วยศาสตราจารย์สถิต ประสมพันธ์

บทที่ 1 บทนำ

ที่มาและความสำคัญ

เพิ่มฝึกฝนการเขียนโปรแกรมตามหลัก Object Oriented Programming ด้วยภาษา Java ในการสร้างเกม

ประเภทโครงงาน

เกม

ประโยชน์

1. เพื่อความสนุกสนาน
2. ฝึกแก้ปัญหาเฉพาะหน้า
3. ฝึกบริหารทรัพยากร

ขอบเขตโครงการ

Sorcerer’s Odyssey

รายละเอียดเกม

ผู้เล่นรับบทเป็นพ่อมดทำภารกิจส่ง คัมภีร์ สู่เมืองที่ห่างไกล ฝ่าฟันอุปสรรค ด้วยดาบและเวทมนต์ เกมจะจบลงเมื่อผู้เล่นพลังชีวิตหมด หรือ ภารกิจสำเร็จ

วิธีการเล่น

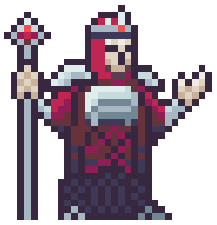
ปุ่ม w , a , s , d ในการควบคุม ขึ้น , ลง , ซ้าย , ขวา ปุ่ม e เพื่อ interact กับ object ในด่าน left click เพื่อ ใช้ดาบ right click เพื่อใช้เวทมนต์

Story board

<u>ตัวละคร</u>

ผู้เล่น

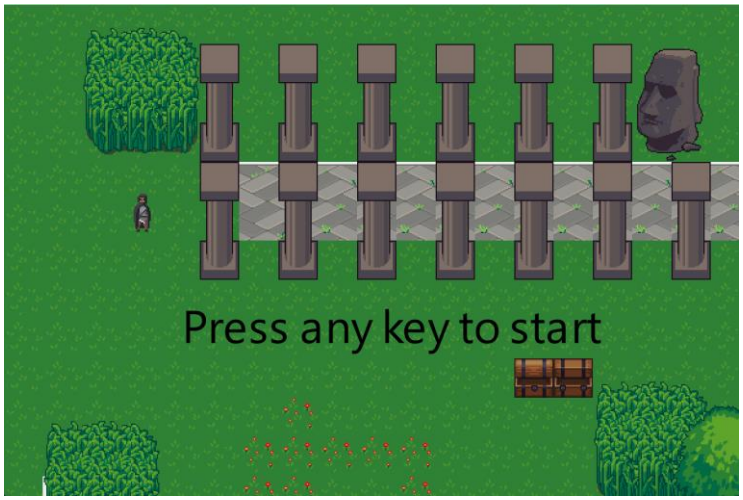
ศัตรู



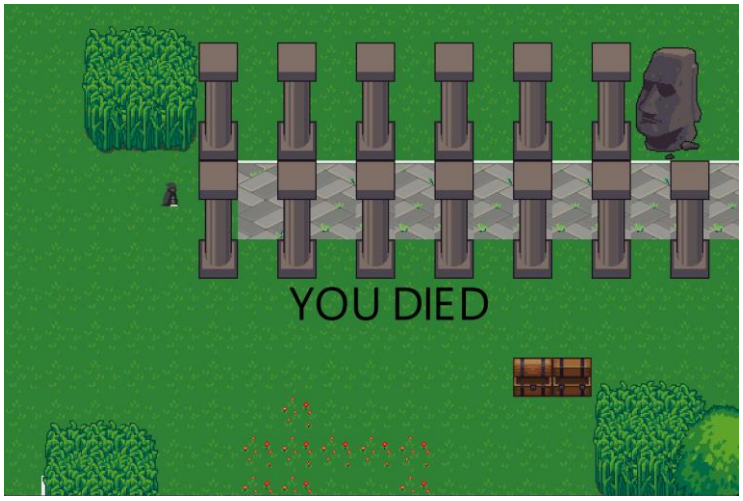
ศัตรูระดับสูง

<u>ฉาก</u>

-เริ่มเกม



-พลังชีวิตหมด game over



-สำเร็จภารกิจ

ตารางแผนการทำงาน

ลำดับ	รายการ	13 ก.ย.-19 ก.ย.	20 ก.ย.-20 ต.ค.	21 ต.ค.-31ต.ค.
1	จัดทำตัวละคร			
2	ศึกษาเอกสารและข้อมูลที่เกี่ยวข้อง			
3	ลงมือเขียนโปรแกรม			
4	จัดทำเอกสาร			
5	ตรวจสอบและแก้ไขข้อผิดพลาด			

บทที่ 2 ส่วนการพัฒนา

2.1 รูปแบบการพัฒนา

2.1.1 สร้าง game loop

2.1.2 สร้างผู้เล่น

2.1.3 สร้างด่าน

2.1.4 สร้างหน้าเมนู

2.1.5 ทำระบบเสียง

2.1.6 สร้างศัตรู

2.1.7 เพิ่มเวทมนต์และ Projectile

2.1.8 เพิ่ม Hud แสดงข้อมูล

2.1.9 สร้าง Event ทำให้มอนสเตอร์เกิดตามเวลาที่กำหนด

2.1.10 สร้างการเปลี่ยนด่าน

2.1.11 สร้างการเก็บของ

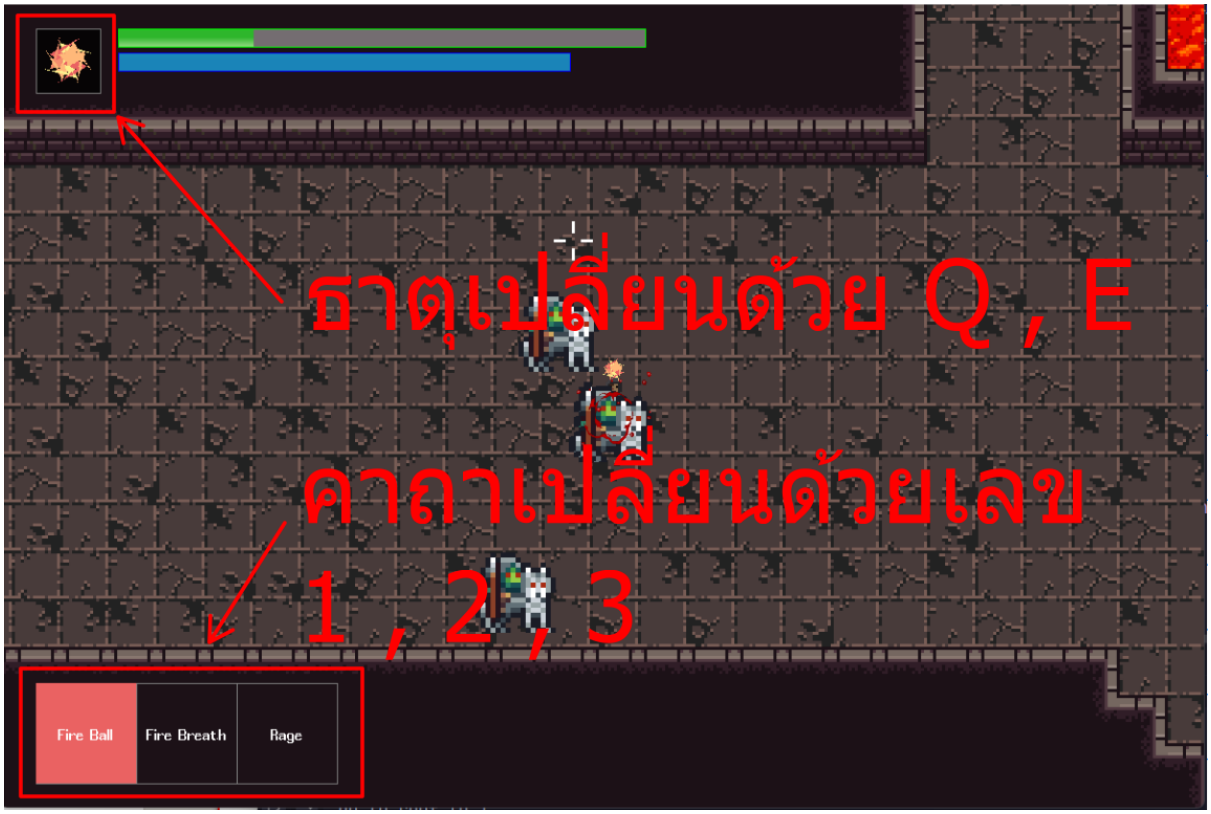
2.2 เนื้อเรื่องย่อ

ผู้เล่นรับบทเป็น พ่อมดจอมขมังเวทย์ที่มีภารกิจในการเดินไปทางไปยังเมืองอันห่างไกล ด้วยเวทย์มนต์เคลื่อนย้าย แต่การใช้เวทย์มนตร์ผิดพลาดทำให้ พ่อมด ตกมาอยู่ใน คุกใต้ดิน ที่เต็มไปด้วยปีศาจร้าย พ่อมด จะหนีออกจาก ภัยร้ายครั้งนี้และทำภารกิจให้สำเร็จได้หรือไม่

2.3 วิธีการเล่น

ผู้เล่นสามารถเคลื่อนที่ด้วยปุ่ม W, A , S ,D

ผู้เล่นจะสามารถใช้เวทย์มนตร์ได้ 4 ธาตุ และ ในแต่ละธาตุจะมีอยู่ 3 คาถา (สามารถ เปลี่ยนธาตุด้วนปุ่ม Q และ E และเลือกคาถาด้วยเลข 1 , 2 ,3)



ภาพ 1การเลือกเวทย์มนต์เบื่องต้น

ในแต่ละด้านจะมี Object ให้ผู้เล่น interact ด้วยปุ่ม F เช่น กล้อง และจุดเคลื่อนย้าย

ภาพ 2Chest

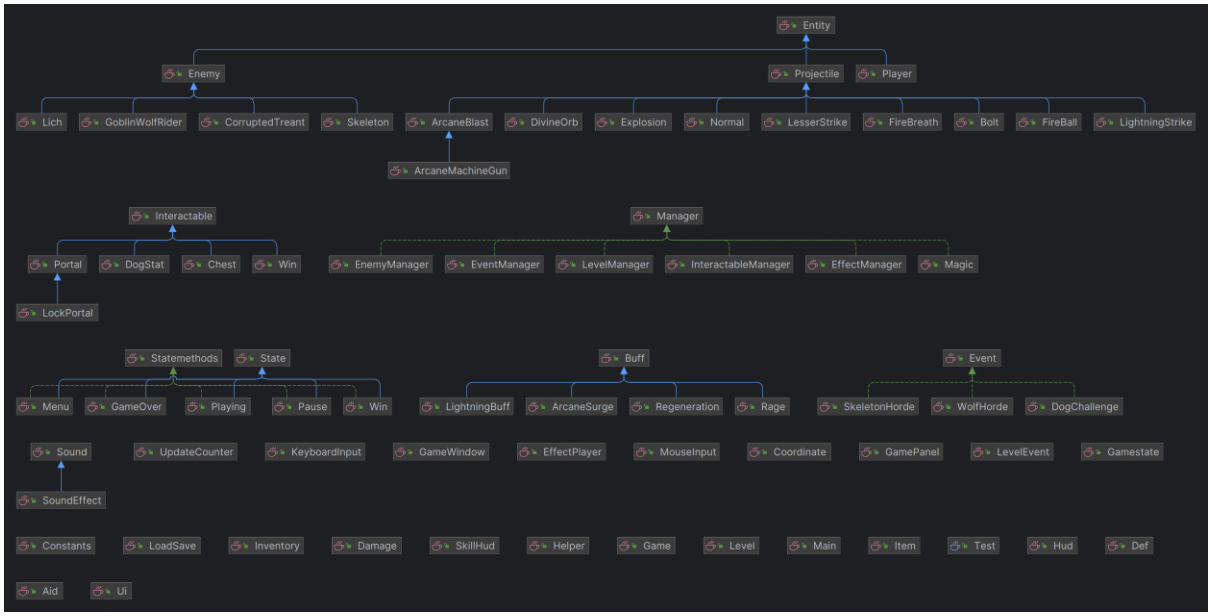
ภาพ 3Protal

Mouse Left จะเป็นการโจมตีระยะประชิด, Mouse Right จะเป็นการใช้ด้วยคาถา. โดยแต่ละคาถาจะมีการใช้ Mana ในการร่ายแตกต่างกัน

ESC เพื่อเข้าหน้าเมนู

2.4 Class Diagram

2.4.1 อย่างย่อ



ภาพ 4Class diagram ย่อ

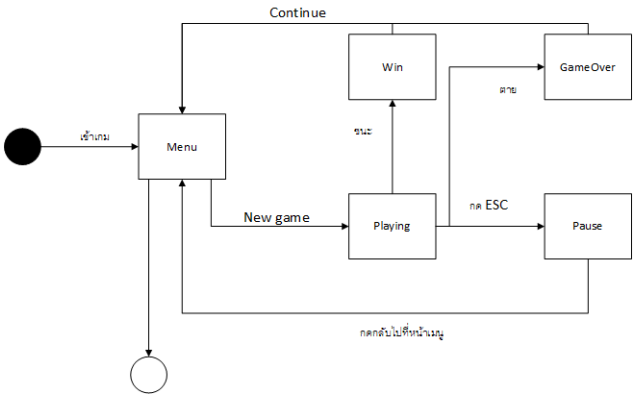
2.4.2 แบบเต็ม

ภาพ 5Class diagram แบบเต็ม

2.5 อธิบายโปรแกรม

เป้าหมายในการเขียน : สร้างเกมที่สามารถขยาย scale ของเกมได้ง่าย

ในเกม มี state ของเกม ประกอบด้วย Menu , Playing ,Pause , GameOver , Win. โดยมีการเปลี่ยน state ดังนี้



ภาพ 6State Diagram

และในแต่ละ State จะมี KeyInput และ MouseInput เป็นของตัวเอง โดยใช้ตัวแปร static ในการเปลี่ยน state

ที่สำคัญที่สุดคือ Playing State ใช้ในการเล่นเกม มีการ update และ วาด Graphics บน panel

2.5.1 Update

จะเรียงลำดับดังนี้

2.5.1.1 Player

2.5.1.2 EnemyManager (**ทำการ** update **ศัตรูทั้งหมดที่อยู่ในด่านเดียวกับผู้เล่น**)

2.5.1.3 EffecftManager (**แสดงเสียงจากคลาส** Sound **และคลาส** Effect **ถูกโจมตีละถูกกำจัด**)

2.5.1.4 Hud (**คำนวณข้อมูลของผู้เล่นเพื่อนำไปทำ** Graphics)

2.5.1.5 Magic (update Projectile **ทั้งหมดที่ผู้เล่นสร้างขึ้น**)

2.5.1.6 EventManager (update **ให้มอนสเตอร์เกิดตามเวลาที่กำหนด**)

2.5.1.7 InteractableManger (**จัดการสิ่งของที่ผู้เล่นสามารถ** interact **ได้**)

```java
@Override
public void update() {
    player.update();

    if (m1pressed) {

        magic.normalAttack(getClickedPos(mousePosX, mousePosY, xLvlOffset,
                yLvlOffset));
    }
    if (m3pressed) {
        if (player.isCastable()) {
            magic.cast(getClickedPos(mousePosX, mousePosY, xLvlOffset,
                    yLvlOffset));
        }

    }
    enemyManager.update();
    effectManager.update();
    hud.update();
    magic.update();
    eventManager.update();
    checkCloseToBorder();
    interactableManager.update();
}
```

ภาพ 7Code การอัปเดตของ Playing

2.5.2 Draw จะเรียงดังนี้ โดยตัวที่อยู่ด้านล่างจะถูกวาดทับตัวที่อยู่ด้านบน.

1. วาดพื้นและกำแพงหลังผู้เล่น
2. วาดผู้เล่น
3. วาดศัตรู
4. วาด effect
5. วาดกำแพงที่อยู่ด้านหน้าผู้เล่น
6. วาดสิ่งที่ผู้เล่นมีปฏิสัมพันธ์ได้ (เป็นกรอบสี่เหลี่ยมมีสีโปร่ง)
7. วาด Projectile และ effect บนหน้าจอของ Magic class
8. วาด Hud แสดงข้อมูลของผู้เล่น

```java
@Override
public void draw(Graphics g) {
    levelManager.drawBehind(g, xLvlOffset, yLvlOffset);
    player.render(g, xLvlOffset, yLvlOffset);
    enemyManager.draw(g, xLvlOffset, yLvlOffset);
    effectManager.draw(g, xLvlOffset, yLvlOffset);
    levelManager.drawFront(g, xLvlOffset, yLvlOffset);
    interactableManager.draw(g, xLvlOffset, yLvlOffset);
    magic.draw(g, xLvlOffset, yLvlOffset);
    int aniIndex = player.getAniIndex();
    g.drawImage(crosshair,  x: mousePosX - (crosshairSize + aniIndex) / 2,  y: mousePosY - (crosshairSize + aniIndex) / 2,
            width: crosshairSize + aniIndex,
            height: crosshairSize + aniIndex,  observer: null);// cross hair
    eventManager.draw(g, xLvlOffset, yLvlOffset);
    hud.draw(g, xLvlOffset, yLvlOffset);

}
```

ภาพ 8Code การวาด Graphics ของ Playing

2.5.3 คลาส Entity ใช้ในการเก็บ hitbox โดยถูกสืบทอดโดยผู้เล่น ศัตรู และ Projectile ต่าง ๆ

```java
12 usages   18 inheritors   SueaWS *
public abstract class Entity {

    protected float width, height;

    protected Rectangle2D.Float hitbox;

    3 usages   SueaWS *
    public Entity(float x, float y, float width, float height) {
        this.width = width;
        this.height = height;
        initHitbox(x, y, width, height);

    }
```

ภาพ 9Code สร้าง hitbox ด้วย Rectangle2D

เป็นคลาสแม่ที่ใช้สร้างศัตรู จะมี Constructor ใช้ในการสร้างพร้อมรับข้อมูลสำคัญ คือ

```
4 usages  SueaWS
public Enemy(int mapIndex, BufferedImage[] animation, float scale, Def def, float x, float y, float width,
             float height) {
    super(x, y, width: scale * width, height: scale * height);
    this.mapIndex = mapIndex;
    this.def = def;
    this.scale = scale;
    this.animation = animation;
    this.isDead = false;
    this.chasing = false;
}
```

ภาพ 10Constructor ของ Enemy

1. mapIndex มีไว้เพื่อบอกว่าเป็นมอนสเตอร์ในด่านไหนเพื่อที่จะไม่ต้อง update หากผู้เล่นไม่ได้อยู่ในด่านนั้น.
2. Animation รับ animation ของมอนสเตอร์เพื่อนำไปวาด โดยถูกโหลดไว้โดย EnemyManger
3. Scale ใช้ในการวาดศัตรูขนาดใด
4. Def ค่าความป้องกันของศัตรู โดยแต่ละตัวจะค่าการป้องกันแตกต่างกัน. (ทำงานตาม ภาพ 11)
5. X และ Y คือตำแหน่งเริ่มต้นของศัตรู
6. Width และ **Height** คือขนาดของ **hitbox** ศัตรู

```
public class Def {

    2 usages
    public double FireDef;
    2 usages
    public double HolyDef;
    2 usages
    public double LightingDef;
    2 usages
    public double ArcaneDef;
    2 usages
    public double PhysicalDef;

    4 usages  SueaWS
    public Def(double fireDef, double holyDef, double lightingDef, double arcaneDef, double physicalDef) {
        FireDef = fireDef;
        HolyDef = holyDef;
        LightingDef = lightingDef;
        ArcaneDef = arcaneDef;
        PhysicalDef = physicalDef;
    }

    1 usage
    public static final Def CorruptedTreantDef = new Def( fireDef: -100, holyDef: -10, lightingDef: 70, arcaneDef: 0, physicalDef: 30);
    1 usage
    public static final Def GoblinWolfRider = new Def( fireDef: 60, holyDef: 50, lightingDef: 0, arcaneDef: 20, physicalDef: 20);
    1 usage
    public static final Def Skeleton = new Def( fireDef: 0, holyDef: -100, lightingDef: 40, arcaneDef: 70, physicalDef: 20);
    1 usage
    public static final Def Lich = new Def( fireDef: 30, holyDef: 40, lightingDef: 50, arcaneDef: 20, physicalDef: 65);
}
```

ภาพ 11Class Def เพื่อเพิ่มพลังป้องกันในศัตรู

2.5.5 การคำนวณความเสียหายของศัตรู

เนื่องจากศัตรูแต่ละประเภทจะมีพลังป้องกัน(เป็น เปอร์เซ็นต์) ต่างกัน ทำให้ต้องมีการคำนวณความเสียหายของ การถูกโจมตีแต่ละครั้งตาม ภาพ 12

```
public void getAttacked(Damage damage) {
    chasing = true;
    switch (damage.getType()) {
        case Damage.FIRE:
            hp -= damage.getDamage() * (100 - def.FireDef) / 100;
            break;
        case Damage.HOLY:
            hp -= damage.getDamage() * (100 - def.HolyDef) / 100;
            break;
        case Damage.LIGHTING:
            hp -= damage.getDamage() * (100 - def.LightingDef) / 100;
            break;
        case Damage.ARCANE:
            hp -= damage.getDamage() * (100 - def.ArcaneDef) / 100;
            break;
        case Damage.PHYSICAL:
            hp -= damage.getDamage() * (100 - def.PhysicalDef) / 100;
            break;
        default:
            break;
    }
    checkDied();
}
```

ภาพ 12Code หักพลังชีวิตตามพลังป้องกัน

จาก Code ข้างต้นทำให้สามารถคำนวณความเสียหายจากการโจมตีได้

2.5.6 ศัตรูโดนโจมตี

จาก 2.5.2 จะเห็นว่า Method getAttacked(Damage damage) จะรับ argument เป็นประเภท Damage ซึ่งมีการเก็บ attribute ตามภาพ 13

```
public class Damage {
    5 usages
    public static final int FIRE = 0;
    2 usages
    public static final int HOLY = 1;
    4 usages
    public static final int LIGHTING = 2;
    3 usages
    public static final int ARCANE = 3;
    2 usages
    public static final int PHYSICAL = 4;
    2 usages
    private final int type;
    3 usages
    private double damage;
    SueaWS
    public Damage(int type, double damage) {
        this.type = type;
        this.damage = damage;
    }
    1 usage   SueaWS
    public int getType() { return type; }
    SueaWS
    public double getDamage() { return damage; }
    SueaWS
    public void setDamage(int damage) { this.damage = damage; }

}
```

ภาพ 13Code Damage Class

Type ของ Damage ถูกทำเป็น int เพื่อให้สามารถงานใช้กับ Switch case ได้ และในแต่ละ คาถาเวทย์มนต์จะมี Damage เป็นของตัวเอง ซึ่งถูกเก็บไว้ คลาสแม่ คือ Class Projectile ตามภาพที่ 14

```
SueaWS *
public Projectile(Damage dmg, Boolean playerOwn, double lifeTime, float xStart, float yStart, Coordinate targetCoor,
        float width,
        float height) {
    super( x: xStart - width / 2,  y: yStart - height / 2, width, height);
    damage = dmg;
    enemyhitted = new ArrayList<>();
```

ภาพ 14Damage ใน class Projectile

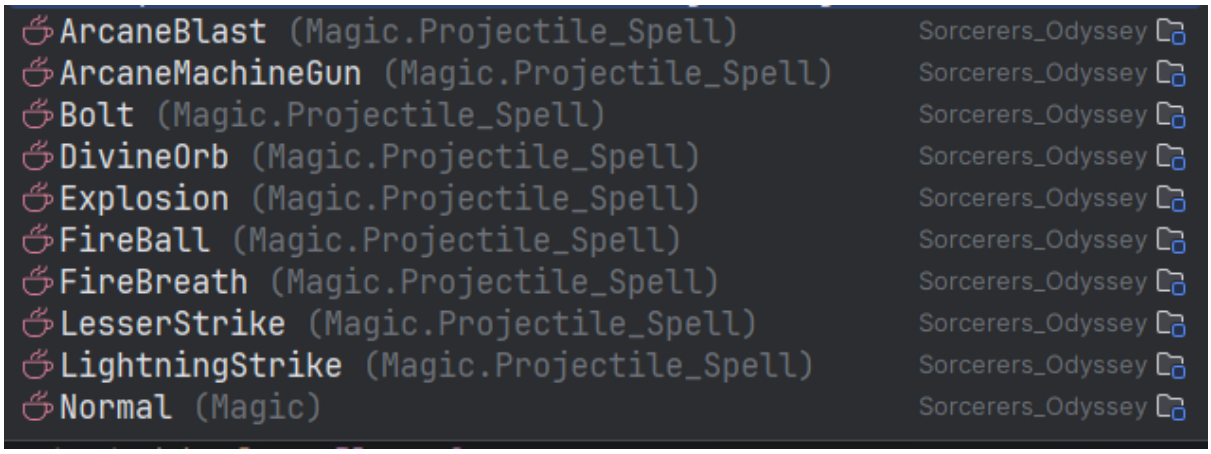
เมื่อ Projectile โดนศัตรูก็จะก็จะเรียก Method hit ของ class Projectile ที่ return damage ของ Projectile นั้น ๆ

```
1 usage  3 overrides  SueaWS
public Damage hit(Entity entity) {
    if (expiredOntarget | expiredOnHit) {
        onExpired();
        active = false;
    }
    onHit();
    return damage;
}
```

ภาพ 15code projectile hit(Entity)

2.5.7 Class Projectile

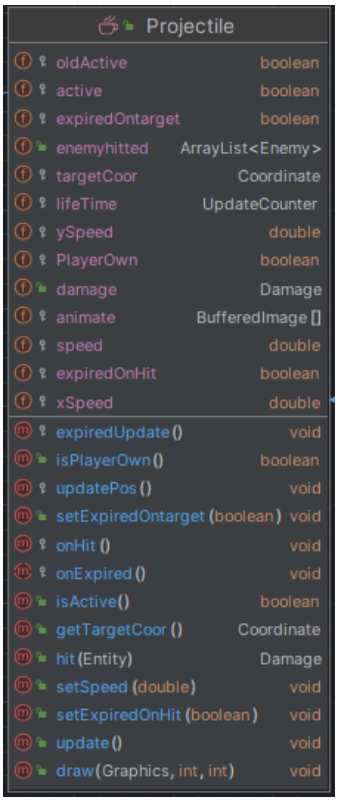
เป็นคลาสที่ถูกใช้ในการสร้าง การโจมตีของผู้เล่น อยู่ 10 อย่าง



ภาพ 16Projectile ถูกใช้ในคลาสดังนี้

โดยภายในของ class Projectile นั้นมี Method และ attribute ที่ทำให้สามารสร้าง การโจมตีที่หลากหลาย และ scale ได้ง่าย เช่น onHIt() onExpired() ที่เอาไว้ให้คลาสลูกสามารถนำไป Override และเลือกได้ว่า Projectile ประเภทที่หายไปเมื่อโดนศัตรู หรือ หายไปเมื่อถึงตำแหน่งที่กำหนด ได้

โดยมีแรงบันดาลใจจากการเขียนภาษา javascript



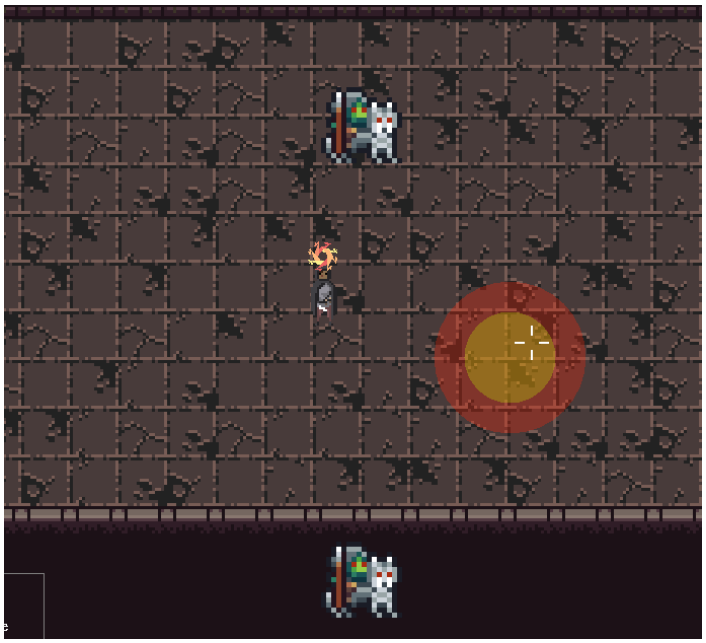
ภาพ 17diagram class Projectile

ตัวอย่าง การใช้ class Projectile ในการสร้าง

```java
public class FireBall extends Projectile {
    public static double cost = 40;
    2 usages
    private SoundEffect castSound;
    2 usages
    private SoundEffect impactSound;
    2 usages
    private Magic magic;

    1 usage  SueaWS
    public FireBall(Magic magic, Coordinate playerCoor, Coordinate targetCoor) {
        super(new Damage(Damage.FIRE,  damage: 80 * Player.dmgMul),  playerOwn: true,  lifeTime: 0.8, playerCoor.x, playerCoor.y, targetCoor,  width: 32,  height: 32);
        setSpeed(9);
        this.magic = magic;
        setExpiredOntarget(true);
        castSound = new SoundEffect( audioPath: "magic/start.wav",  volPercent: 70);
        impactSound = new SoundEffect( audioPath: "magic/end.wav",  volPercent: 70);
        castSound.play();
    }

    SueaWS
    @Override
    protected void onExpired() {
        impactSound.play();
        Coordinate curCoor = new Coordinate(getCenterX(), getCenterY());
        magic.castExplosion(curCoor);
    }

}
```

ภาพ 18ตัวอย่างการใช้ class projectile class FireBall

การภาพจะเห็นว่า class FireBall เป็นคลาสที่ มี life time 0.8 วินาที ขนาด 32x32 pixel สร้างความเสียหาย 80 ประเภทไฟ มีความเร็ว 9 และ จะหายไปเมื่อถึงเป้าหมายที่กำหนด และเมื่อ FireBall หายไป จะเรียก Method onExpired() ที่ถูก Override ทำให้สามารถเล่นเสียงระเบิด และ สร้างระเบิดลูกเล็ก ตามภาพที่ 19



ภาพ 19onExpired() สร้างระบิด

2.5.8 วิธีเช็คว่า Projectile โจมตีโดน

จาก Projectile และ Enemy มี hitbox เหมือนกันทำให้สามารถเรียก Method intersects(Rectangle2D r) เพื่อเช็คได้ แต่เนื่องจาก Projectile สามารถ intersects กับหลายครั้ง ทำให้ในแต่ละ Projectile ต้องมี ArrayList เพื่อเก็บศัตรูที่โดนโจมตีไปแล้ว

```
for (int j = 0; j < magic.projectiles.size(); j++) {
    Projectile projectile = magic.projectiles.get(j);
    if (!projectile.enemyhitted.contains(enemy) && enemy.getHitbox().intersects(projectile.getHitbox()) && projectile.isPlayerOwn()) {
        enemy.getAttacked(projectile.hit(enemy));
        effectManager.playAttacked(enemyCenterX, enemyCenterY,  isPlayer: false);
        projectile.enemyhitted.add(enemy);
    }
}
```

ภาพ 20วิธีเช็ค Projectile intersect กับ Enemy

2.5.9 เวทย์มนตร์

Class Magic เป็นคลาสที่ทำให้ผู้เล่นสามารถใช้เวทมนตร์ได้ โดยในการเลือก เวทมนตร์ที่จะใช้ทำได้โดยการใช้ตัวแปรประเภท static

```
ublic class Magic implements Manager {

    8 usages
    public static final int Fire = 0;
    7 usages
    public static final int Arcane = 1;
    7 usages
    public static final int Lightning = 2;
    7 usages
    public static final int Holy = 3;

    15 usages
    public static int selectedElement = Magic.Fire;
    8 usages
    public static int selectedChoice = 0;
```

ภาพ 21ใช้ในการเลือกธาตุและเวทย์มนตร์ที่จะใช้

หลังจากที่มีตัวแปรแล้วเราสามารถนำไปสร้าง method cast(Coordinate targetCoor) ได้ภายใน method จะมี switch case nested อยู่ ทำให้สามารถเลือก ธาตุและลำดับท่าได้ ตามภาพ 22

```
public void cast(Coordinate targetCoor) {
        switch (selectedElement) {
            case Fire:
                switch (selectedChoice) {
                    case 0:
                        castFireBall(targetCoor);
                        break;
                    case 1:
                        castFireBreath(targetCoor);
                        break;
                    case 2:
                        castRage();
                        break;
                }
                break;
            case Arcane:
                switch (selectedChoice) {
                    case 0:
                        castArcaneBullets(targetCoor);
                        break;
                    case 1:
                        castArcaneMachineGun(targetCoor);
                        break;
                    case 2:
                        castArcaneSurge();
                        break;
                }
                break;
            case Lightning:
                switch (selectedChoice) {
                    case 0:
                        castBolt(targetCoor);
                        break;
                    case 1:
                        castLightningStrike(targetCoor);
                        break;
                    case 2:
                        castLightningBuff();
                        break;
                }
                break;
            case Holy:
                switch (selectedChoice) {
                    case 0:
                        castDivineOrb(targetCoor);
                        break;
                    case 1:
                        castDivineAid();
                        break;
                    case 2:
                        castRegeneration();
                        break;
                }
                break;

        }
    }
```

ภาพ 22เลือก คาถาที่จะร่าย

Method ที่ชื่อว่า cast… เป็น method ที่ทำหน้าที่คำนวณ cost ของ คาถา และนำเข้าไปเก็บใน ArrayList<Projectile> projectiles มีการทำ Polymorphism ให้สามารถเก็บ เวทประเภท Projectile ได้ เพื่อนำไป update และ draw ต่อไป

```
public ArrayList<Projectile> projectiles = new ArrayList<>();
```

ภาพ 23projectiles Polymorphism

รวมถึง คาถาประเภท เสริมพลังผู้เล่นที่ ArrayList<Buff> buffs ที่เก็บค่าเอาไว้ แบบ Polymorphism

```
public ArrayList<Buff> buffs = new ArrayList<>();
```

ภาพ 24biff Polymorphism

ตัวอย่าง Method ที่เมื่อผู้เล่นคาถา

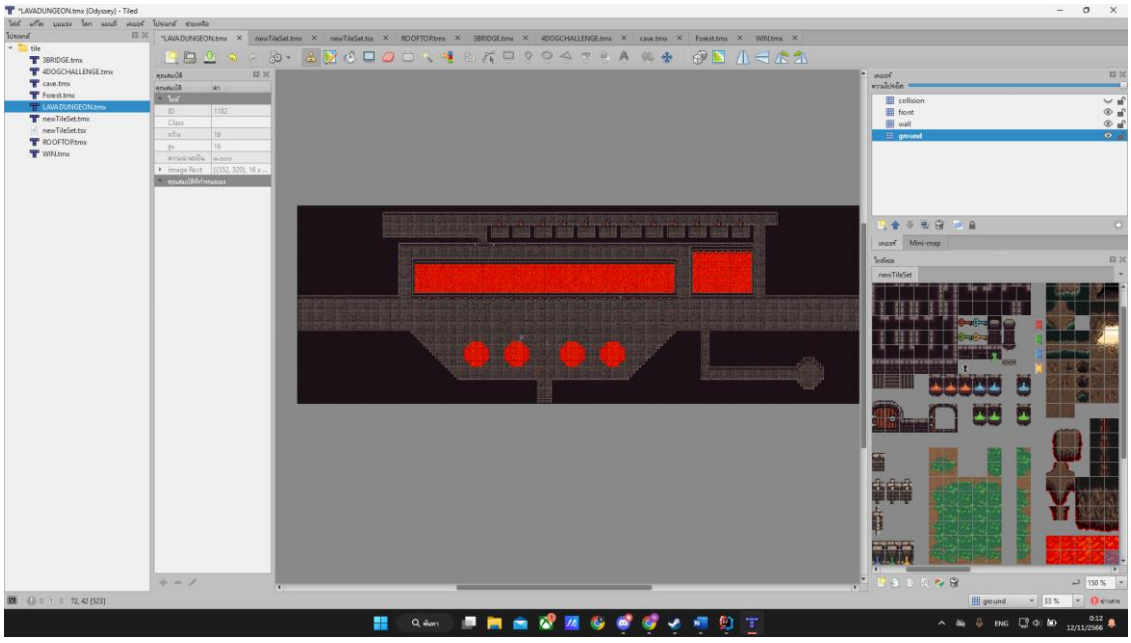
```
public void castFireBall(Coordinate targetCoor) {
    if (player.castSpell(FireBall.cost)) {
        projectiles.add(new FireBall( magic: this, player.getPlayerCenter(), targetCoor));
    }
}
```

ภาพ 25ร่ายคาถา

หลังจาก castFireBall ถูกเรียกจะทำการเช็คกับ class Player ว่าสามารถร่าย คาถาได้หรือไม่ผ่าน Method castSpell(double cost):Boolean ของ object Player หากผู้เล่นสามารถร่ายคาถาได้ก็จะเพิ่ม Object FireBall ที่มุ่งสู่ targetCoor เข้าสู่ projectiles เพื่อ update และ draw ต่อไป

2.5.10 **สร้างด่าน**

ในเกมจะประกอบไปด้วย 7 ด่านถูกทำในโปรแกรม Tiled (https://www.mapeditor.org/) ตามภาพ 26



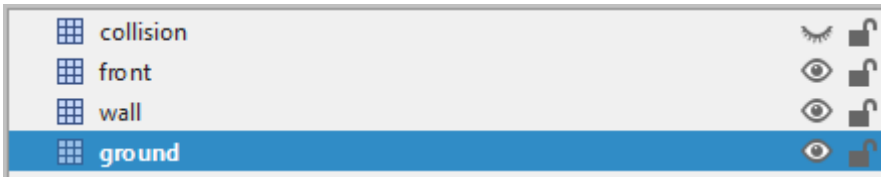
*ภาพ 26ตัวอย่าง โปรแกรม **Tiled***

ในเกมประกอบไปด้วย 7 ด่าน

1. Lava Dungeon
2. Roof top
3. Bridge
4. Challenge
5. Forest
6. Cave
7. Win

โปรแกรม tiled จะสามารถเอา tile set มีมาเพื่อวาดเป็นด่านได้ขนาด tile ละเท่า ๆ กันโดยใน แต่ละด่านจะแบ่งเป็น 4 Layer

1. Ground
2. Wall
3. Front
4. Collision

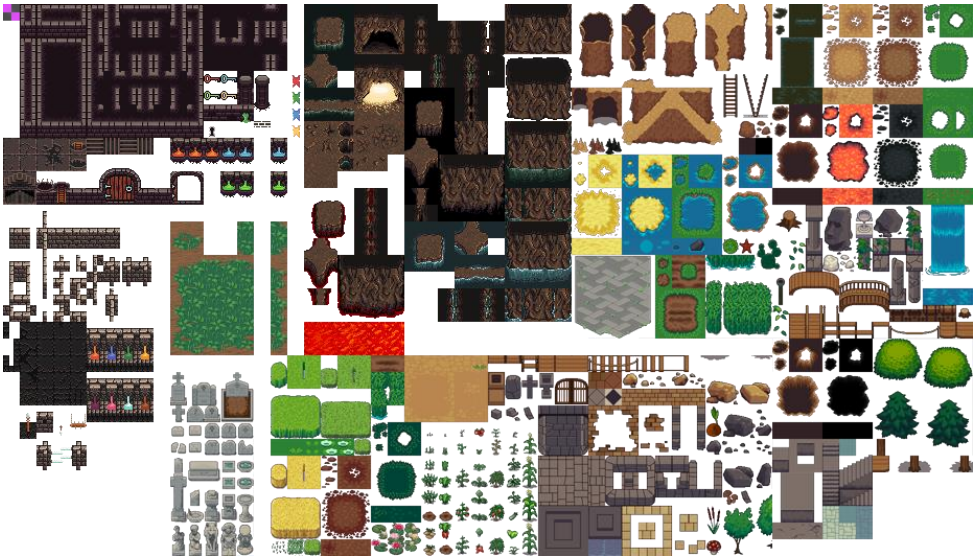


ภาพ 27Layer ของแต่ละด่าน

หลังจากวาดเสร็จแล้วสามารถ export สิ่งที่วาดไว้เป็นไฟล์ csv ได้ในแต่ละช่องที่วาดจะเก็บ ID ของ tile ไว้ จากนั้นนำไฟล์ที่ได้ไปอ่านและเก็บเป็นประเภท int[][] เพื่อใช้ในการวาดเป็น Graphics ต่อไป

2.5.11 Graphics ของด่าน

หลังจากที่ได้ของมูลของ Tile ต่าง ๆ ใน ด่านเป็น int[][] แล้วก็จำเป็นต้อง โหลด tile set เข้ามาสำหรับการทำ Graphics หลังจากนั้นจะใช้ method getSubimage เพื่อทำให้ในแต่ละตำแหน่งของ int[][] และมีการคำนวณ index เพิ่มให้เหมือนกับ CSV ที่ได้มาจาก Program Tiled



ภาพ 28tile set ที่ใช้ในเกม

```
private void draw(Graphics g, int layer, int xLvlOffset, int yLvlOffset) {
    for (int j = 0; j < levelLayers[layer].getYlength(); j++) {
        for (int i = 0; i < levelLayers[layer].getXlength(); i++) {
            int index = levelLayers[layer].getSpriteIndex(i, j);
            if (index >= 0) {
                g.drawImage(levelSprite[index],
                        x: (int) (i * TILE_SIZE) - xLvlOffset,
                        y: (int) (j * TILE_SIZE) - yLvlOffset,
                        TILE_SIZE,
                        TILE_SIZE,
                        observer: null);
            }
        }
    }
}
```

ภาพ 29code การวาด ด่านเป็น Layer

2.5.12 Event

ในเกมจะมีการสร้าง event ที่ทำให้ศัตรูเกิดตามเวลาที่กำหนด (ในบางด่าน) เพื่อความท้าทายของเกม โดยมีการสร้าง inner class ที่มีการ extends class UpdateCounter ที่ทำหน้าที่ update เมื่อถึงเวลาที่กำหนด

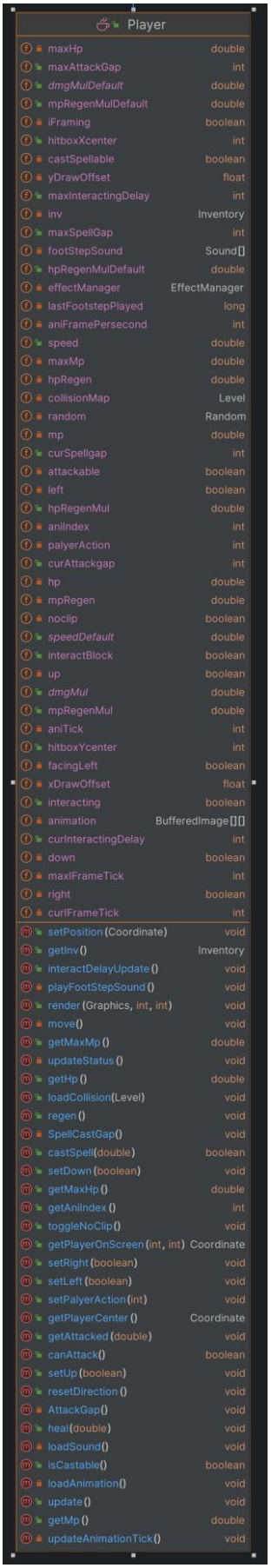
```
private class HordeSpawn extends UpdateCounter {

    // EnemyManager em;
    4 usages
    private final Playing playing;

    1 usage  SueaWS
    public HordeSpawn(Playing playing, double second, boolean cycle) {
        super(second, cycle);
        this.playing = playing;
    }

    1 usage  SueaWS
    @Override
    public void onUpdate() {
        Coordinate playerCoor = playing.getPlayer().getPlayerCenter();
        playing.getEnemyManager().spawnSkeletion(mapIndex,  x: playerCoor.x - 400, playerCoor.y,  aggro: true);
        playing.getEnemyManager().spawnSkeletion(mapIndex,  x: playerCoor.x + 400, playerCoor.y,  aggro: true);
    }

}
```

ภาพ 30 inner class ใน SkeletonHorde

คลาส UpdateCounter เป็นคลาสที่ถูกใช้ใน คลาส Projectile, Buff, Event ต่าง ๆ รวมถึงคลาสจำพวกที่ต้องทำสิ่งนั้น ๆ ตามเวลาที่กำหนดไว้ เพื่อแก้ปัญหาที่ต้องใช้ตัวแปรเยอะที่เกิดขึ้นใน class Player ยกตัวอย่างการคำนวณระยะห่างของการโจมตีของผู้เล่น จำเป็นต้องใช้ตัวแปรอย่างน้อย 3 ตัวคือ attackable curAttackgap และ maxAttackGap

```
private void AttackGap() {
    if (!attackable) {
        curAttackgap++;
        if (curAttackgap >= maxAttackGap) {
            curAttackgap = 0;
            attackable = true;
        }
    }
}
```

ภาพ 31ระยะห่างการโจมีตีของผู้เล่น



*ภาพ 32ปัญหา **class Player** มีตัวแปรเยอะ*

แต่เนื่องจาก class Player เป็น class ที่ละเอียดอ่อนทำให้การที่สร้าง UpdateCounter ขึ้นมาทีหลังไม่สามารถนำไปใช้ได้

2.5.13 Level Event

ในแต่ละด่านสามารถมีการเกิด Event ที่แตกต่างกันได้เมื่อเข้าด่านเช่น ด่านที่ 1 Lava dungeon เป็นด่านที่มีศัตรูขี่หมาป่า เกิดขึ้นมา ส่วนนี้จะมีการให้ Level Manager เรียกใช้ Level Event ซึ่งเป็น interface ที่มีmethod on Enter และ on Exit ตามภาพ 33 โดยสร้างเป็น array ของ interface Level Event

```
public interface LevelEvent {

    1 usage  7 implementations  SueaWS
    public void onEnter();

    1 usage  7 implementations  SueaWS
    public void onExit();
}
```

ภาพ 33code interface Level Event

```
levelEvents[LAVADUNGEON] = new LevelEvent() {

    1 usage  SueaWS
    @Override
    public void onEnter() { playing.geteventManager().wolfHordeStart(LAVADUNGEON); }

    1 usage  SueaWS
    @Override
    public void onExit() { playing.geteventManager().wolfHordEnd(LAVADUNGEON); }

};
```

ภาพ 34ตัวอย่างการเพิ่ม Level Event ใน class Level Manager

จากภาพที่ 34 ทำให้เห็นว่าเมื่อเข้าสู่ด่าน Lava Dungeon จะมีการสั่งให้ Event Manager เริ่ม Event เรียกฝูงปีศาจเข้ามาโจมตีผู้เล่น และจะจบ Event เมื่อผู้เล่นออกจากด่าน

2.5.14 abstract class Interactable

เกมจำเป็นต้องสิ่งของที่ผู้เล่นสามารถมีปฏิสัมพันธ์ได้ โดย abstract class Interactable มีไว้เพื่อทำสิ่งนั้น โดยมีเป้าหมายที่จะทำให้เกมสามารถ scale ขนาดขึ้นไปได้เรื่อย ๆ

```
public abstract class Interactable {
    11 usages
    protected String massage;
    protected Rectangle2D.Float hitbox;
    14 usages
    protected boolean showMessage = false;
    no usages  SueaWS
    public boolean isShowMessage() { return showMessage; }
    2 usages  SueaWS
    public void setShowMessage(boolean showMessage) { this.showMessage = showMessage; }
    SueaWS
    public Rectangle2D.Float getHitbox() { return hitbox; }
    4 usages  SueaWS
    public Interactable(String massage) {
        hitbox = new Rectangle2D.Float( x: 0f,  y: 0f,  w: 0f,  h: 0f);
        this.massage = massage;
    }
    1 usage  SueaWS
    public void interact() { onSubmit(); }
    1 usage  5 implementations  SueaWS
    public abstract void onSubmit();
    4 implementations  SueaWS
    public abstract void draw(Graphics g, int xLvlOffset, int yLvlOffset);
    5 implementations  SueaWS
    public abstract void update();
    no usages  4 implementations  SueaWS
    public abstract void onIntersects();
    no usages  4 implementations  SueaWS
    public abstract void notIntersects();
}
```

ภาพ 35code abstract class Interactable

ตัวอย่าง การใช้ abstract class Interactable ในการสร้าง class กล่อง

```
1 usage   SueaWS
@Override
public void onSubmit() {
    if (!taked) {
        playing.getPlayer().getInv().add(item);
    }
    taked = true;
}
```

ภาพ 36code onSubmit ของ Chest

ทำงานโดยเพิ่มของตาม id เข้าสู่ช่องเก็บของผู้เล่นและวาดตำแหน่งที่สามารถมีปฏิสัมพันธ์กับกล่องได้ รวมถึงสร้าง Graphics แสดงว่า กล่องนั้นทำอะไรได้

```
@Override
public void draw(Graphics g, int xLvlOffset, int yLvlOffset) {
    if (!taked) {
        g.setColor(new Color( r: 0,  g: 9,  b: 250,  a: 30));
        g.fillRect((int) (hitbox.x - xLvlOffset), (int) (hitbox.y - yLvlOffset), (int) (hitbox.width),
                (int) (hitbox.height));

        if (showMessage) {
            g.setFont(font);
            Coordinate coor = Ui.GetTextMiddleScreen(g, massage);
            Graphics2D g2d = (Graphics2D) g;

            int yStart = Ui.GetPercentY(60);
            Point start = new Point( x: Config.SCREEN_WIDTH / 2, yStart);
            Point end = new Point( x: Config.SCREEN_WIDTH / 2, Config.SCREEN_HEIGHT);

            GradientPaint gradientPaint = new GradientPaint(start, new Color( r: 0,  g: 0,  b: 0,  a: 0), end, new Color( r: 0,  g: 10,  b: 40));
            g2d.setPaint(gradientPaint);
            g2d.fillRect( x: 0, start.y, SCREEN_WIDTH,  height: Config.SCREEN_HEIGHT - start.y);

            g.setColor(Color.white);
            g.drawString(massage, coor.x, Ui.GetPercentY(80));
        }
    }
}
```

ภาพ 37การสร้าง Graphics ของกล่อง

ภาพ 38กรอบที่สามารถมี ปฏิสัมพันธ์กับกล่อง และ Graphics แสดง action เมื่อมีปฏิสัมพันธ์กับกล่อง

2.5.15 เปลี่ยนด่าน

ในการเคลื่อนที่ไปแต่ละด่านภายในเกมสามารถใช้ประตูเคลื่อนย้าย ซึ่งเป็น class ลูก ของ Interactable ที่ Override Method onSubmit ที่ทำให้ผู้เล่นย้ายไปยังตำแหน่ง ๆ ในด่าอื่นหรือในด่านเดียวกันได้ โดยการวาดตำแหน่งที่สามารถมีปฏิสัมพันธ์จะเป็นพื้นที่สีม่วง

```
1 usage   1 override   SueaWS *
@Override
public void onSubmit() {
    playing.goTomap(targetMap);
    System.out.println(LevelManager.curMapIndex);
    playing.getPlayer().setPosition(targetCoor);
    playing.getPlayer().interacting = false;
    playing.getPlayer().interactBlock = true;
}
```

ภาพ 39onSubmit ของ class Portal

2.5.16 Constructor

```
2 usages   SueaWS
public EffectPlayer(BufferedImage[] animation, int x, int y, double scale) {
    this.animation = animation;
    this.x = x;
    this.y = y;
    this.scale = scale;

}
```

ภาพ 40 ตัวอย่าง Constructorของ Effect Player

ตัวอย่าง ของโปรแกรมที่มี Constructor ในภาพคือ Constructor ของ class EffectPlayer มีหน้าที่เล่น animation ศัตรูหรือผู้เล่นถูกโจมตีหรือตาย

2.5.17 Encapsulation

```
public class Sound {
    4 usages
    protected URL url;
    19 usages
    protected Clip clip;
    7 usages
    protected FloatControl volumeControl;

    5 usages   SueaWS
    public Sound(String audioPath) {
        this.url = getClass().getResource( name: "/res/asset/sound/" + audioPath);
        loadAudio();
    }

    1 usage   SueaWS *
    private void loadAudio() {
        try {
            AudioInputStream audioInputStream = AudioSystem.getAudioInputStream(url);
            clip = AudioSystem.getClip();
            clip.open(audioInputStream);
            volumeControl = (FloatControl) clip.getControl(FloatControl.Type.MASTER_GAIN);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

ภาพ 41 Sound Encapsulation

ตัวอย่าง ของโปรแกรมที่มทำ Encapsulation ในภาพคือ attribute และ method ของคลาส Sound ทำให้ที่ในการโหลดและเล่นเสียงเมื่อถึงสร้างตัวอย่างการใช้เช่น การเล่นเสียงเพลงในหน้า Menu โดย attribute url , clip, volumeControl จะมีการเอาไปใช้ในคลาสลูก class SoundEffect และ private method loadAudio ใช้ภายใน class ไม่มีการนำไปใช้ที่อื่น

2.5.18 Composition

```
public class Player extends Entity {


    2 usages
    private final EffectManager effectManager;
```

ภาพ 42 composition ใน Player

ตัวอย่าง ของโปรแกรมที่มีทำ Composition ในภาพคือ การที่ class Player ที่มี Class EffectManager อยู่ด้านในเพื่อให้ผู้เล่นสามารถเล่น Effect Player โดนโจมตีและตายได้

2.5.19 Polymorphism

```
public class Magic implements Manager {

    18 usages
    public ArrayList<Projectile> projectiles = new ArrayList<>();
    12 usages
    public ArrayList<Buff> buffs = new ArrayList<>();
    8 usages
```

ภาพ 43 Polymorphism

จะมีการสร้าง ArrayList เอาไว้เก็บตัว Object ประเภท Projectile เมื่อมีการ new Object เข้าไปจะเป็น Classลูกของ Projectile เช่น FireBall , Bolt , FireBreath

```
projectiles.add(new FireBall( magic: this, player.getPlayerCenter(), targetCoor));
```

ภาพ 44 เพิ่ม Fireball ลงใน Projectiles

```
projectiles.add(new Bolt( magic: this, player.getPlayerCenter(), targetCoor));
```

ภาพ 45 เพิ่ม bolt l ลงใน Projectiles

```
projectiles.add(new FireBreath(player.getPlayerCenter(), targetCoor));
```

ภาพ 46เพิ่ม Fire Breath ลงใน Projectiles

2.5.20 Abstract

```
public abstract class Projectile extends Entity {
```

ภาพ 47 abstract class Projectile

จากภาพที่ 47 จะเห็นว่า class Projectile เป็น abstract class ที่สามารถมี abstract method ได้เช่น onExpird()

```
protected abstract void onExpired();
```

ภาพ 48 abstract method

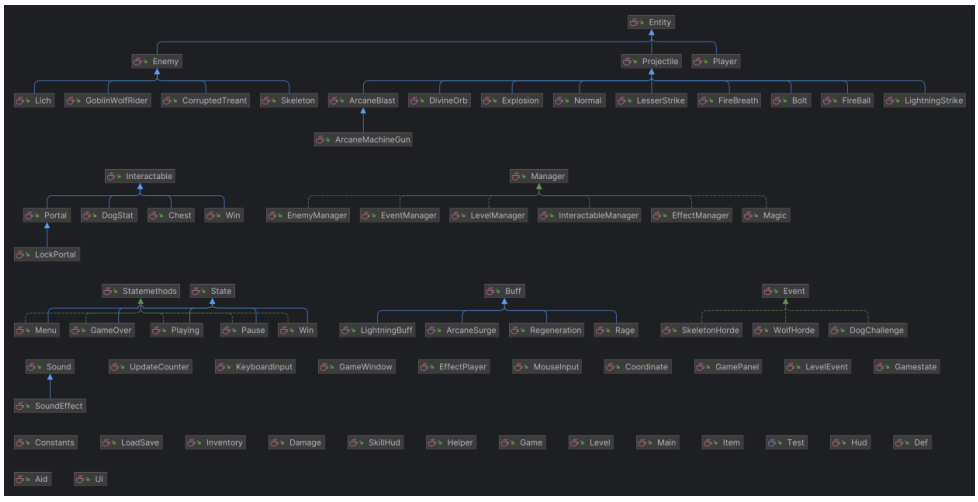
เมื่อมีการ Inherit class Projectile ไปจำเป็นต้อง Define body ของ onExpired() ด้วยเช่น class LightningStrike ที่เป็น class ลูกของ Projectile จะ Define การทำงานของ onExpired() ดังนี้

```
@Override
protected void onExpired() {
    Coordinate top = new Coordinate((int) (targetCoor.x + width), (int) (targetCoor.y + 120 + width));
    Coordinate down = new Coordinate((int) (targetCoor.x + width), (int) (targetCoor.y - 120 + width));
    Coordinate left = new Coordinate((int) (targetCoor.x - 120 + width), (int) (targetCoor.y + width));
    Coordinate right = new Coordinate((int) (targetCoor.x + 120 + width), (int) (targetCoor.y + width));
    magic.spawnLesserStrike(top);
    magic.spawnLesserStrike(down);
    magic.spawnLesserStrike(left);
    magic.spawnLesserStrike(right);
}
```

ภาพ 49 Override abstract method

2.5.20 Inheritance

มีการสืบทอดตาม class diagram



2.5.21 **โครงสร้าง** GUI

ประกอบไปด้วย

1. JFrame
2. JPanel

จะมีการวาดทุกอย่างเป็น Graphics ใน JPanel

2.5.22 Event handling

ในแต่ละ game state จะมี MouseInput และ KeyInput เป็นของตัวเอง

บทที่ 3 สรุป

3.1 ปัญหาที่พบระหว่างการพัฒนา

เป็นปัญหาที่เจอระหว่างการพัฒนา ส่วนใหญ่นั้นแก้ไขไปแล้ว

3.1.1 constructor ยาวเกินไป

3.1.2 ไฟล์ด่านทั้ง 4 Layer โหลดผิดพลาด (แก้แล้ว)

3.1.3 การวาดด่านที่มีขนาดเล็กกว่าหน้าจอไม่ได้

3.1.4 ไม่สามารถใช้เสียงที่มากกว่า 16 bits ได้ (แก้แล้ว)

3.1.5 export jar file แล้วไม่มีเสียง (แก้แล้ว)

3.1.6 ปัญหา performance เมื่อด่านมีขนาดใหญ่

3.1.7 jar file ไม่สามารถทำงานได้เมื่อไม่อยู่ใน dir ของเกม (แก้แล้ว)

3.1.8 ประตูเคลื่อนย้ายทำงานผิดพลาดเมื่ออยู่ใน jar file (แก้แล้ว)

3.2 จุดเด่นของโปรแกรม

3.2.1 มีเวทมนต์ให้ใช้ทั้งหมด 12 ท่า

3.2.2 มีด่านมากถึง 7 ด่า

3.2.3 เกมมีความท้าทาย ไม่เร็วหรือช้าจนเกินไป

3.2.4 ศัตรูมีพลังป้องกันที่แตกต่างกัน ทำให้ผู้เล่นต้องปรับตัวเพื่อเอาชนะ

3.2.5 สามารถเพิ่ม scale ของเกมได้ง่าย

3.2.6 มีการโหลดด่านเป็นไฟล์ CSV ทำให้สามารถใช้ โปรแกรม TILED ช่วยในการสร้างได้

3.2.7 แต่ละด่านแบ่งเป็น 4 Layer ทำให้การสร้างด่าน สามารถทำได้ละเอียด

3.2.8 เป็นเกมที่มีเสียง