Mini Project

This Cake IS MINE

Submitted by

Pornpipat Utamawuttikumjon Student ID 6304062660047

Present to

Asst. Prof. Sathit Prasomphan

This report is part of the course 040613204

Object-Oriented Programming section 5th

Department of Computer and Information Science

Faculty of Applied Science

King Mongkut's University of Technology North Bangkok

Semester 1 Academic year 2023

Unit 1

➳ Introduction :

This is my mini project improve my skill computers programing by java language and use OOP concept and give more than ever bot in principles and including causal logic skills.

➳ Type project : A shooter-defense, 2D pixel art

➳ Benefits :

1) Practice tact and reflexes
2) Fun and stress relief
3) Hand-Eyes Coordination
4) Entertainment and Enjoyment

➳ Proposal : Proposal This Cake IS MINE

➳ Work Schedule

Order	List	1-8	9-13	13-23	24-31
1	Make character and design				
2	Research and Idea Generation				
3	Programming				
4	Documentation				
5	Testing and find Error				

Unit 2

➳ Game details:

"This Cake IS MINE", You take on the role of a little girl who wants to protect her cake from enemies who want to eat it.

You must take down your enemies, While the enemies will come continuously. until time run out.

➳ How to play :

Use **the mouse** to point at an enemy and click left click to fire various bullets at enemies until time over or if you miss your shot, enemies can touch your cake, it decrease your Cake Health until reach 0 and you will lose.

➳ Character

Little girl

Slime(Enemy)

Cake

➳ Detail of Program:

```
public ThisCakeIsMineGame() {
    loadBackgroundImages();
    initializeComponents();
    addMouseListeners();
    setPreferredSize(new Dimension(width: PANEL_WIDTH, height: PANEL_HEIGHT));
    gameState = GameState.MENU;
    addMouseListener(l: this);
}

private void loadBackgroundImages() {
    try {
        backgroundImage = ImageIO.read(input: getClass().getResource(name: "BackgroundFix.png"));
        backgroundMenuImage = ImageIO.read(input: getClass().getResource(name: "BackgroundMenu.png"));
        tryAgainButtonImage = ImageIO.read(input: getClass().getResource(name: "TryAgainButtonImage.png"));
        exitButtonImage = ImageIO.read(input: getClass().getResource(name: "ExitButtonImage.png"));
        menuImage = ImageIO.read(input: getClass().getResource(name: "MENU.png"));

        tryAgainButtonX = (PANEL_WIDTH - tryAgainButtonImage.getWidth()) / 2;
        tryAgainButtonY = 375;
        exitButtonX = (PANEL_WIDTH - exitButtonImage.getWidth()) / 2;
        exitButtonY = 450;
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

```
private void initializeComponents() {
    player = new Player();
    cake = new Cake();
    enemies = new ArrayList<>();
    bullets = new ArrayList<>();
    cakeHealth = CAKE_INITIAL_HEALTH;
}

public void startGame1() {
```

```
private void startTimer() {
    gameTimer = new Timer(delay: 1000, (ActionEvent e) -> {
        countdownTime--;
        if (countdownTime <= 0) {
            // Handle game over or end of countdown here
            gameState = GameState.MENU;
            countdownTime = 0; // Reset the countdown time
        }
        repaint();
    });
    gameTimer.start();
}

private void stopGame() {
    if (gameTimer != null) {
        gameTimer.stop();
    }
}
```

```
public void gameOver() {
    gameState = GameState.GAME_OVER;
    stopGame(); // Stop the game loop
    loadBackgroundImages(); // Load the "BackgroundGameOver.png" image
    repaint();
}
```

➳ Encapsulation:

```
public class ThisCakeIsMineGame extends JPanel implements MouseMotionListener, MouseListener {
    private static final int PANEL_WIDTH = 1600;
    private static final int PANEL_HEIGHT = 900;
    private static final int ENEMY_SPAWN_RATE = 3;
    private static final int CAKE_INITIAL_HEALTH = 100;

    private BufferedImage backgroundMenuImage;
    private Image backgroundImage;
    private Player player;
    private Cake cake;
    private ArrayList<Enemy> enemies;
    private ArrayList<Bullet> bullets;
    private Point mousePosition;
    private int countdownTime;
    private Timer gameTimer;
    private int cakeHealth;
    private Image menuImage;
    private BufferedImage tryAgainButtonImage;
    private BufferedImage exitButtonImage;
    private int tryAgainButtonX;
    private int tryAgainButtonY;
    private int exitButtonX;
    private int exitButtonY;
    private GameState gameState;

    private enum GameState {
        MENU,
        GAME1,
        GAME_OVER,
    }
```

```
public class Bullet {
    private double x;
    private double y;
    private double velocityX;
    private double velocityY;
    private static final double SPEED = 10;
```

```
public class Entity {
    protected int x;
    protected int y;
    protected BufferedImage image;

    public void draw(Graphics g) {
        g.drawImage(img:image, x, y, observer:null);
```

```
public class Player extends Entity {
    private boolean isShooting;
    private long lastFireTime;
    private long frameDelay = 1000 / 120; // 120 FPS
    private BufferedImage bulletImage;
    private long lastFrameTime; // Add this field
    private long bulletDelay = 500;

    private int lastFrameX;
    private int lastFrameY;
```

```
public class Cake extends Entity {
    public Cake() {
        loadImage();
        x = 250;
        y = 410;
    }
```

```
public class Enemy extends Entity {
    private int targetX;
    private int targetY;
    private int speed;
    private int yDirection;
    private int directionChangeInterval;
    private long lastDirectionChangeTime = System.currentTimeMillis();

    // Define the y-range limits
    private static final int Y_MIN = 150;
    private static final int Y_MAX = 650;
```

➳ Composition:

```
private void initializeComponents() {
    player = new Player();
    cake = new Cake();
    enemies = new ArrayList<>();
    bullets = new ArrayList<>();
    cakeHealth = CAKE_INITIAL_HEALTH;
}
```

```
public class Player extends Entity {
    private boolean isShooting;
    private long lastFireTime;
    private long frameDelay = 1000 / 120; // 120 FPS
    private BufferedImage bulletImage;
    private long lastFrameTime; // Add this field
    private long bulletDelay = 500;
```

```
private void loadImage() {
    try {
        BufferedImage originalPlayerImage = ImageIO.read(input: getClass().getResource(name: "Player_One.png"));
        int scaledWidth = originalPlayerImage.getWidth() * 2;
        int scaledHeight = originalPlayerImage.getHeight() * 2;
        image = new BufferedImage(width: scaledWidth, height: scaledHeight, imageType: BufferedImage.TYPE_INT_ARGB);
        Graphics2D g2dPlayer = image.createGraphics();
        g2dPlayer.drawImage(img:originalPlayerImage, x: 0, y: 0, width: scaledWidth, height: scaledHeight, observer: null);
        g2dPlayer.dispose();

        bulletImage = ImageIO.read(input: getClass().getResource(name: "bulletplayer.png"));
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

```
}
```

```
public BufferedImage getBulletImage() {
    return bulletImage;
}
```

```java
public class Enemy extends Entity {
    private int targetX;
    private int targetY;
    private int speed;
    private int yDirection;
    private int directionChangeInterval;
    private long lastDirectionChangeTime = System.currentTimeMillis();
```

```java
    private void loadImage() {
        try {
            BufferedImage originalEnemyImage = ImageIO.read(input: getClass().getResource(name: "ModelEnemy1.png"));
            int scaledEnemyWidth = originalEnemyImage.getWidth() * 2;
            int scaledEnemyHeight = originalEnemyImage.getHeight() * 2;
            image = new BufferedImage(width: scaledEnemyWidth, height: scaledEnemyHeight, imageType: BufferedImage.TYPE_INT_ARGB);
            Graphics2D g2dEnemy = image.createGraphics();
            g2dEnemy.drawImage(img:originalEnemyImage, x: 0, y: 0, width: scaledEnemyWidth, height: scaledEnemyHeight, observer: null);
            g2dEnemy.dispose();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

➳ Polymorphism :

```java
public class Entity {
    protected int x;
    protected int y;
    protected BufferedImage image;

    public void draw(Graphics g) {
        g.drawImage(img:image, x, y, observer: null);
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }
}
```

```java
public class Player extends Entity {
    private boolean isShooting;
    private long lastFireTime;
    private long frameDelay = 1000 / 120; // 120 FPS
    private BufferedImage bulletImage;
    private long lastFrameTime; // Add this field
    private long bulletDelay = 500;
```

```java
public class Cake extends Entity {
    public Cake() {
        loadImage();
        x = 250;
        y = 410;
    }

}

public class Enemy extends Entity {
    private int targetX;
    private int targetY;
    private int speed;
    private int yDirection;
```

➵ Abstract :

```java
public abstract class Entity {
    protected int x;
    protected int y;
    protected BufferedImage image;

    public void draw(Graphics g) {
        g.drawImage(img:image, x, y, observer:null);
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }
}
```

➳ Inheritance :

Player Class:

```
    public long getLastFrameTime() {
        return lastFrameTime; // Return the new field
    }

    public void setLastFrameTime(long lastFrameTime) {
        this.lastFrameTime = lastFrameTime; // Set the new field
    }

    public long getFrameDelay() {
        return frameDelay;
    }

    public BufferedImage getBulletImage() {
        return bulletImage;
    }
}
```

Cake Class:

```
    @Override
    public int getX() {
        return x;
    }

    @Override
    public int getY() {
        return y;
    }
```

Enemy Class:

```
    }
    @Override
    public int getX() {
        return x;
    }

    @Override
    public int getY() {
        return y;
    }
}
```

➳ GUI :

```
public class ThisCakeIsMineGame extends JPanel implements MouseMotionListener, MouseListener {
    private static final int PANEL_WIDTH = 1600;
    private static final int PANEL_HEIGHT = 900;
    private static final int ENEMY_SPAWN_RATE = 3;
    private static final int CAKE_INITIAL_HEALTH = 100;

    private BufferedImage backgroundMenuImage;
    private Image backgroundImage;
    private Player player;
    private Cake cake;
    private ArrayList<Enemy> enemies;
    private ArrayList<Bullet> bullets;
    private Point mousePosition;
    private int countdownTime;
    private Timer gameTimer;
    private int cakeHealth;
    private Image menuImage;
    private BufferedImage tryAgainButtonImage;
    private BufferedImage exitButtonImage;
    private int tryAgainButtonX;
    private int tryAgainButtonY;
    private int exitButtonX;
    private int exitButtonY;
    private GameState gameState;
```

```
    private void loadBackgroundImages() {
        try {
            backgroundImage = ImageIO.read(input: getClass().getResource(name: "BackgroundFix.png"));
            backgroundMenuImage = ImageIO.read(input: getClass().getResource(name: "BackgroundMenu.png"));
            tryAgainButtonImage = ImageIO.read(input: getClass().getResource(name: "TryAgainButtonImage.png"));
            exitButtonImage = ImageIO.read(input: getClass().getResource(name: "ExitButtonImage.png"));
            menuImage = ImageIO.read(input: getClass().getResource(name: "MENU.png"));

            tryAgainButtonX = (PANEL_WIDTH - tryAgainButtonImage.getWidth()) / 2;
            tryAgainButtonY = 375;
            exitButtonX = (PANEL_WIDTH - exitButtonImage.getWidth()) / 2;
            exitButtonY = 450;
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

```
    private void initializeComponents() {
        player = new Player();
        cake = new Cake();
        enemies = new ArrayList<>();
        bullets = new ArrayList<>();
        cakeHealth = CAKE_INITIAL_HEALTH;
    }
```

```
    public ThisCakeIsMineGame() {
        loadBackgroundImages();
        initializeComponents();
        addMouseListeners();
        setPreferredSize(new Dimension(width: PANEL_WIDTH, height: PANEL_HEIGHT));
        gameState = GameState.MENU;
        addMouseListener(l: this);
    }
```

```java
@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);

    // Draw the semi-transparent background image (BackgroundMenu.png) first
    if (gameState == GameState.MENU) {
        g.drawImage(img:backgroundImage, x: 0, y: 0, width: PANEL_WIDTH, height: PANEL_HEIGHT, observer: this);

        player.draw(g);
        cake.draw(g);

        Graphics2D g2d = (Graphics2D) g;
        g2d.setComposite(comp: AlphaComposite.getInstance(rule: AlphaComposite.SRC_OVER, alpha: 0.6f));
        g2d.drawImage(img:backgroundMenuImage, x: 0, y: 0, width: PANEL_WIDTH, height: PANEL_HEIGHT, observer: this);
        g2d.setComposite(comp: AlphaComposite.getInstance(rule: AlphaComposite.SRC_OVER, alpha: 1.0f)); // Reset opacity

        g.drawImage(img:menuImage, x: 0, y: 0, width: PANEL_WIDTH, height: PANEL_HEIGHT, observer: this);
    } else if (gameState == GameState.GAME1) {
        // Draw the "BackgroundFix.png" as the game background
        g.drawImage(img:backgroundImage, x: 0, y: 0, width: PANEL_WIDTH, height: PANEL_HEIGHT, observer: this);

        // Continue drawing other game elements in the foreground
        player.draw(g);
        cake.draw(g);

        for (Enemy enemy : enemies) {
            enemy.draw(g);
        }

        for (Bullet bullet : bullets) {
            bullet.draw(g);
        }

        drawCountdownTimer(g);
        drawCakeHealth(g);
    } else if (gameState == GameState.GAME_OVER) {
        g.drawImage(img:backgroundImage, x: 0, y: 0, width: PANEL_WIDTH, height: PANEL_HEIGHT, observer: this);

        player.draw(g);
        cake.draw(g);

        for (Enemy enemy : enemies) {
            enemy.draw(g);
        }

        for (Bullet bullet : bullets) {
            bullet.draw(g);
        }

        drawCountdownTimer(g);
        drawCakeHealth(g);

        // Now, draw the "BackgroundGameOver.png" with 60% opacity over the existing game elements
```

```
        drawCountdownTimer(g);
        drawCakeHealth(g);
    } else if (gameState == GameState.GAME_OVER) {
        g.drawImage(img:backgroundImage, x: 0, y: 0, width: PANEL_WIDTH, height: PANEL_HEIGHT, observer: this);

        player.draw(g);
        cake.draw(g);

        for (Enemy enemy : enemies) {
            enemy.draw(g);
        }

        for (Bullet bullet : bullets) {
            bullet.draw(g);
        }

        drawCountdownTimer(g);
        drawCakeHealth(g);

        // Now, draw the "BackgroundGameOver.png" with 60% opacity over the existing game elements
        Graphics2D g2d = (Graphics2D) g;
        g2d.setComposite(comp: AlphaComposite.getInstance(rule: AlphaComposite.SRC_OVER, alpha: 0.6f));
        g2d.drawImage(img:backgroundMenuImage, x: 0, y: 0, width: PANEL_WIDTH, height: PANEL_HEIGHT, observer: this);
        g2d.setComposite(comp: AlphaComposite.getInstance(rule: AlphaComposite.SRC_OVER, alpha: 1.0f)); // Reset opacity

        drawButtons(g);
    }
}
```

```
private void drawButtons(Graphics g) {

    g.drawImage(img:tryAgainButtonImage, x: tryAgainButtonX, y: tryAgainButtonY, observer: this);
    g.drawImage(img:exitButtonImage, x: exitButtonX, y: exitButtonY, observer: this);

}
```

```
private void drawCountdownTimer(Graphics g) {
    g.setColor(c: Color.WHITE);
    g.setFont(new Font(name: "Pixeboy", style: Font.PLAIN, size: 48));

    int minutes = countdownTime / 60;
    int seconds = countdownTime % 60;

    String timerText = String.format(format: "%02d:%02d", args: minutes, args: seconds);
    int textWidth = g.getFontMetrics().stringWidth(str:timerText);
    int x = 750;
    int y = 50;

    g.drawString(str:timerText, x, y);
}
```

```java
private void drawCakeHealth(Graphics g) {
    g.setColor(c: Color.WHITE);
    g.setFont(new Font(name: "Pixeboy", style: Font.PLAIN, size: 48));

    // Load the cake icon image
    ImageIcon icon = new ImageIcon(location: getClass().getResource(name: "Cake_Icon.png"));
    Image iconImage = icon.getImage();

    // The icon
    int iconX = 30;
    int iconY = 40 - iconImage.getHeight(observer: null) / 2;

    // Define icon
    g.drawImage(img:iconImage, x: iconX, y: iconY, observer: null);

    // Text
    String healthText = "Cake HP: " + cakeHealth + "/100";
    int textWidth = g.getFontMetrics().stringWidth(str:healthText);
    int x = iconX + iconImage.getWidth(observer: null) + 10;
    int y = 50;

    [illegible]

    // Load the cake icon image
    ImageIcon icon = new ImageIcon(location: getClass().getResource(name: "Cake_Icon.png"));
    Image iconImage = icon.getImage();

    // The icon
    int iconX = 30;
    int iconY = 40 - iconImage.getHeight(observer: null) / 2;

    // Define icon
    g.drawImage(img:iconImage, x: iconX, y: iconY, observer: null);

    // Text
    String healthText = "Cake HP: " + cakeHealth + "/100";
    int textWidth = g.getFontMetrics().stringWidth(str:healthText);
    int x = iconX + iconImage.getWidth(observer: null) + 10;
    int y = 50;

    g.drawString(str:healthText, x, y);
}

private void loadMenuImage() {
    try {
        menuImage = ImageIO.read(input: getClass().getResource(name: "MENU.png"));
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

```
    ,

    private void loadImage() {
        try {
            BufferedImage originalEnemyImage = ImageIO.read(input: getClass().getResource(name: "ModelEnemy1.png"));
            int scaledEnemyWidth = originalEnemyImage.getWidth() * 2;
            int scaledEnemyHeight = originalEnemyImage.getHeight() * 2;
            image = new BufferedImage(width: scaledEnemyWidth, height: scaledEnemyHeight, imageType: BufferedImage.TYPE_INT_ARGB);
            Graphics2D g2dEnemy = image.createGraphics();
            g2dEnemy.drawImage(img:originalEnemyImage, x: 0, y: 0, width: scaledEnemyWidth, height: scaledEnemyHeight, observer: null);
            g2dEnemy.dispose();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
    ,

    private void loadImage() {
        try {
            BufferedImage originalCakeImage = ImageIO.read(input: getClass().getResource(name: "Cake_Object.png"));
            int scaledCakeWidth = originalCakeImage.getWidth() * 3;
            int scaledCakeHeight = originalCakeImage.getHeight() * 3;
            image = new BufferedImage(width: scaledCakeWidth, height: scaledCakeHeight, imageType: BufferedImage.TYPE_INT_ARGB);
            Graphics2D g2dCake = image.createGraphics();
            g2dCake.drawImage(img:originalCakeImage, x: 0, y: 0, width: scaledCakeWidth, height: scaledCakeHeight, observer: null);
            g2dCake.dispose();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    @Override
```

```
    private void loadImage() {
        try {
            BufferedImage originalPlayerImage = ImageIO.read(input: getClass().getResource(name: "Player_One.png"));
            int scaledWidth = originalPlayerImage.getWidth() * 2;
            int scaledHeight = originalPlayerImage.getHeight() * 2;
            image = new BufferedImage(width: scaledWidth, height: scaledHeight, imageType: BufferedImage.TYPE_INT_ARGB);
            Graphics2D g2dPlayer = image.createGraphics();
            g2dPlayer.drawImage(img:originalPlayerImage, x: 0, y: 0, width: scaledWidth, height: scaledHeight, observer: null);
            g2dPlayer.dispose();

            bulletImage = ImageIO.read(input: getClass().getResource(name: "bulletplayer.png"));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

➳ Event Handling:

```
private GameState gameState;

private enum GameState {
    MENU,
    GAME1,
    GAME_OVER,
}

public ThisCakeIsMineGame() {
    loadBackgroundImages();
    initializeComponents();
    addMouseListeners();
    setPreferredSize(new Dimension(width: PANEL_WIDTH, height: PANEL_HEIGHT));
    gameState = GameState.MENU;
    addMouseListener(l: this);
}
```

```
@Override
public void mouseClicked(MouseEvent e) {
    if (gameState == GameState.MENU) {
        startGame1();
        requestFocusInWindow(); // Request focus to handle key events
        startGameLoop(); // Start the game loop
    } else if (gameState == GameState.GAME_OVER) {

        // Check if the Try Again button is clicked
        if (e.getX() >= tryAgainButtonX && e.getX() <= tryAgainButtonX + tryAg
                && e.getY() >= tryAgainButtonY && e.getY() <= tryAgainButtonY
            // Restart the game (you should add the logic for this)
            restartGame(); // You need to implement this method
        }

        // Check if the Exit button is clicked
        if (e.getX() >= exitButtonX && e.getX() <= exitButtonX + exitButtonIma
                && e.getY() >= exitButtonY && e.getY() <= exitButtonY + exitBu
            // Exit the game (you should add the logic for this)
            System.exit(status: 0); // Exit the game
        }
    }
}
```

```java
@Override
public void mousePressed(MouseEvent e) {
    if (e.getButton() == MouseEvent.BUTTON1) {
        player.setShooting(shooting: true);
    }
}

@Override
public void mouseReleased(MouseEvent e) {
    if (e.getButton() == MouseEvent.BUTTON1) {
        player.setShooting(shooting: false);
    }
}

@Override
public void mouseEntered(MouseEvent e) {
    // Not needed, but required by MouseListener
}

@Override
public void mouseExited(MouseEvent e) {
    // Not needed, but required by MouseListener
}
```

```java
private void startGameLoop() {
    Timer timer = new Timer(delay: 0, (ActionEvent e) -> {
        long currentTime = System.currentTimeMillis();
        if (currentTime - player.getLastFrameTime() > player.getFrameDelay()) {
            player.setLastFrameTime(lastFrameTime: currentTime);
            updateGame();
            repaint();
        }
    });
    timer.setRepeats(flag: true);
    timer.setDelay(delay: 0);
    timer.start();
}

private void restartGame() {
    // Restart the game when needed
    gameState = GameState.GAME1;
    countdownTime = 60;
    startTimer();
    cakeHealth = CAKE_INITIAL_HEALTH;
    initializeComponents();
    startGameLoop();
}
```

➳ Algorithm:

```
public void move() {
    double distanceToCake = Math.hypot(targetX - x, targetY - y);

    if (distanceToCake < 1) {
        enemies.remove(o: this);
        return;
    }

    if (distanceToCake < 600) {
        targetX = cake.getX();
        double angle = Math.atan2(cake.getY() - y, cake.getX() - x);
        x += speed * Math.cos(a: angle);
        y += speed * Math.sin(a: angle);
    } else {
        long currentTime = System.currentTimeMillis();
        if (currentTime - lastDirectionChangeTime > directionChangeInterval) {
            lastDirectionChangeTime = currentTime;
            yDirection = new Random().nextInt(bound: 3) - 1;
            directionChangeInterval = new Random().nextInt(bound: 3000) + 1000;
        }

        y = Math.max(a: Y_MIN, b: Math.min(a: Y_MAX, b: y));

        double angle = Math.atan2(targetY - y, targetX - x);
        x += speed * Math.cos(a: angle);
        y += speed * yDirection;
    }

}

}

public boolean hasCollidedWithBullet(Bullet bullet) {
    double distanceToBullet = Math.hypot(bullet.x - x, bullet.y - y);
    return distanceToBullet < 20; // Adjust collision radius as needed
}

@Override
```

```
public void shoot(ArrayList<Bullet> bullets, Point mousePosition) {
    long currentTime = System.currentTimeMillis();
    if (currentTime - lastFireTime > frameDelay) {
        lastFireTime = currentTime;

        double dx = mousePosition.getX() - (x + image.getWidth() / 2);
        double dy = mousePosition.getY() - (y + image.getHeight() / 2);
        double angle = Math.atan2(y: dy, x: dx);

        double velocityX = Math.cos(a: angle) * Bullet.SPEED;
        double velocityY = Math.sin(a: angle) * Bullet.SPEED;

        bullets.add(new Bullet(x + image.getWidth() / 2, y + image.getHeight() / 2, velocityX, velocityY));
    }
}
```

Unit 3

➳ **Problems encountered during development :**

1. Not according to plan.
2. The work didn’t turn out as expected because of limited time and insufficient knowledge, Including learning that cannot be learned in time because of short time.

➳ **Highlight of Program**

It is a game genre that is not commonly explored and features the dynamic creation of various enemies through spawning, contributing to a more engaging and interesting gameplay experience.

➳ **Suggestions for teachers who want to explain or that I did not understand after studying Or would you like to add more for the next generation?**

อยากให้รุ่นน้องบางคนเรียนเท่าที่ไหวครับ มีอะไรก็ถามอาจารย์ได้ เพราะบางครั้งเรียนหนักไปแล้วสมองบางคนจะรับไม่ค่อยไหว