VIETNAM INTERNATIONAL UNIVERSITY
HO CHI MINH CITY

INTERNATIONAL UNIVERSITY
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

OBJECT-ORIENTED PROGRAMMING

Final Project

Game: LAST TIME ADVENTURE

BY GROUP - Last Time

https://github.com/nguyeenphan/LastTimeAdventure.git

Contents

CHAPTER 1: INTRODUCTION

1.1 Gaming in the Field:

The video game industry has seen exponential growth over the past few decades, evolving from simple, pixelated experiences to complex, immersive worlds. One of the most cherished genres that has maintained its popularity is the wave-based survival game. These games challenge players to endure increasingly difficult waves of enemies, testing their strategic planning and quick reflexes. The appeal lies in the escalating difficulty, the thrill of survival, and the gratification of overcoming relentless adversaries. This genre's success can be attributed to its ability to provide both short, intense gameplay sessions and long-term strategic planning, making it accessible yet deeply engaging.

1.2 About the game project:

"Last Time Adventure" is a capstone project developed by a dedicated team of students passionate about retro gaming. Our goal is to create a 2D top-down wave-based survival game that pays homage to the classic 8-bit era while incorporating modern gameplay mechanics. The project involves extensive collaboration, combining our skills in programming, art design, and game theory. Throughout the development process, we focused on creating a balanced and challenging experience that captures the essence of classic arcade games, while introducing innovative elements to keep players engaged.

1.3 Our game:

In "Last Time Adventure," players find themselves in a vibrant, pixelated world teeming with dangerous creatures. As the last hero standing, they must survive endless waves of enemies, each more challenging than the last. The game features intuitive controls and a variety of enemy types that require different strategies to defeat. Players must strategically position themselves to outlast the onslaught. The retro art style and chiptune soundtrack immerse players in a nostalgic yet fresh gaming experience. "Last Time Adventure" is designed to be easy to pick up but hard to master, providing endless replay-ability for both casual gamers and hardcore enthusiasts.

1.4 References:

- https://stackoverflow.com/
- https://www.youtube.com/
- https://chatgpt.com/

1.5 Developer team:

Name- Github Username	ID	Contributes
Vũ Thành An -Nope ITITIU21148	ITITIU21148	Write report, uml, Ppt, Make sound.
Nguyễn Hoàng Minh Nghĩa - nghianguyen19	ITITIU21255	Make GUI, Write report, make health bar of characters.
Nguyễn Hà Thanh -ZackNguyen0211	ITITIU21144	Make map, make images of tiles for map , make game mechanics, sound effects, debug.
Phan Nguyễn Khánh Nguyên -nguyeenphan	ITCSIU21209	Make monsters and images of characters and monsters.

CHAPTER 2: SOFTWARE REQUIREMENTS

2.1 What we have:

- Basic and friendly system for people to enjoy the game in order to learn the fundamentals of basic coding.
- Minimum maintenance cost (graphics).
- Easy to operate.
- Only available on PC
- With measured coding and professional thinking.

2.2 What we want:

- Develop a system within limited cost.
- Maximum high definition.
- Design the whole system in an efficient manner.
- .Easy to update.
- Adding more features including game modes, function, quality of life and supporting more platforms (Android, iOS, Web3,...)

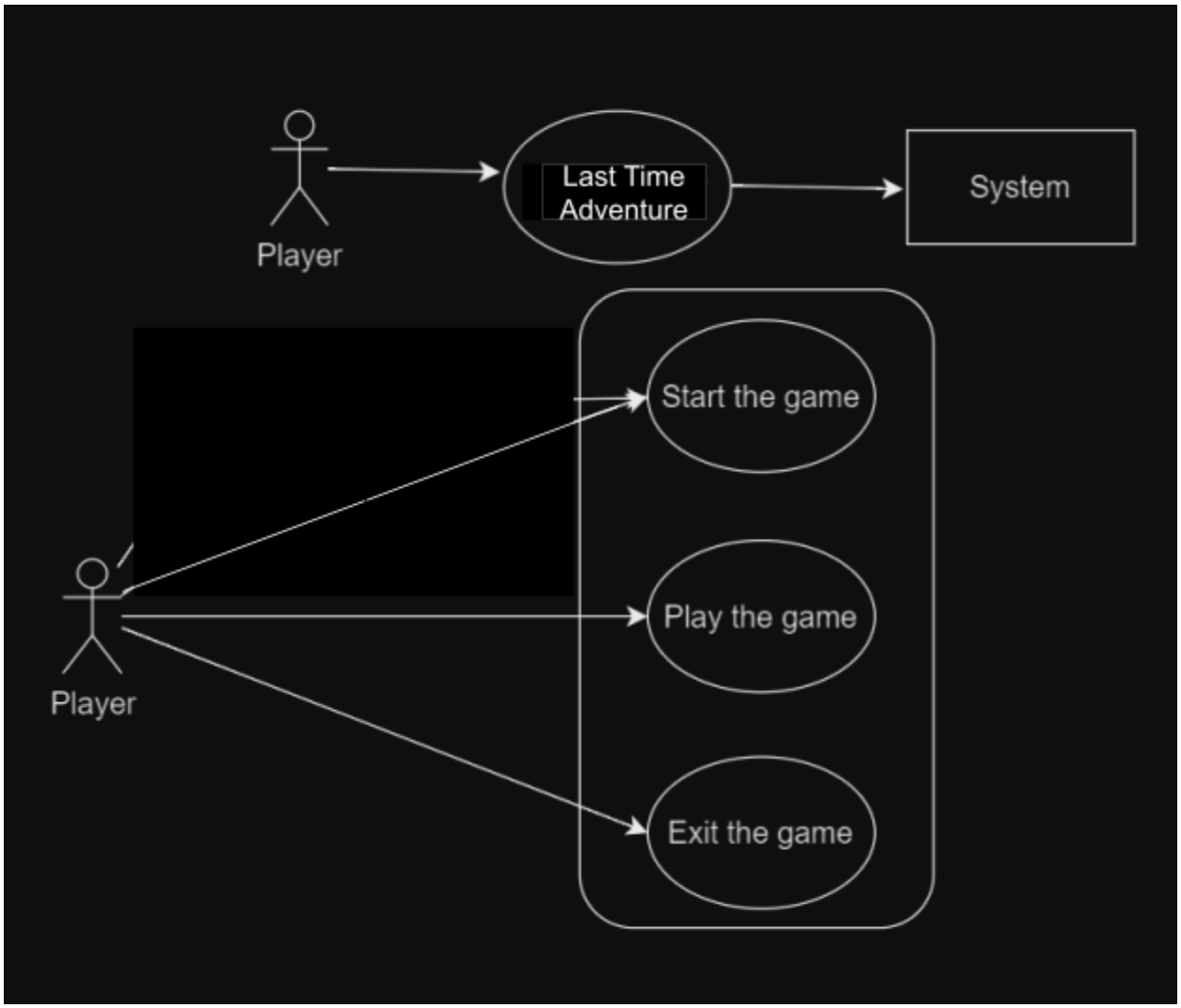
2.3 Working tools and platform

- IntelliJ(JDK 22 Library included)
- Visual Studio Code(JDK 22 Library included)
- GitHub
- Pixilart

2.4 Use Case Scenario

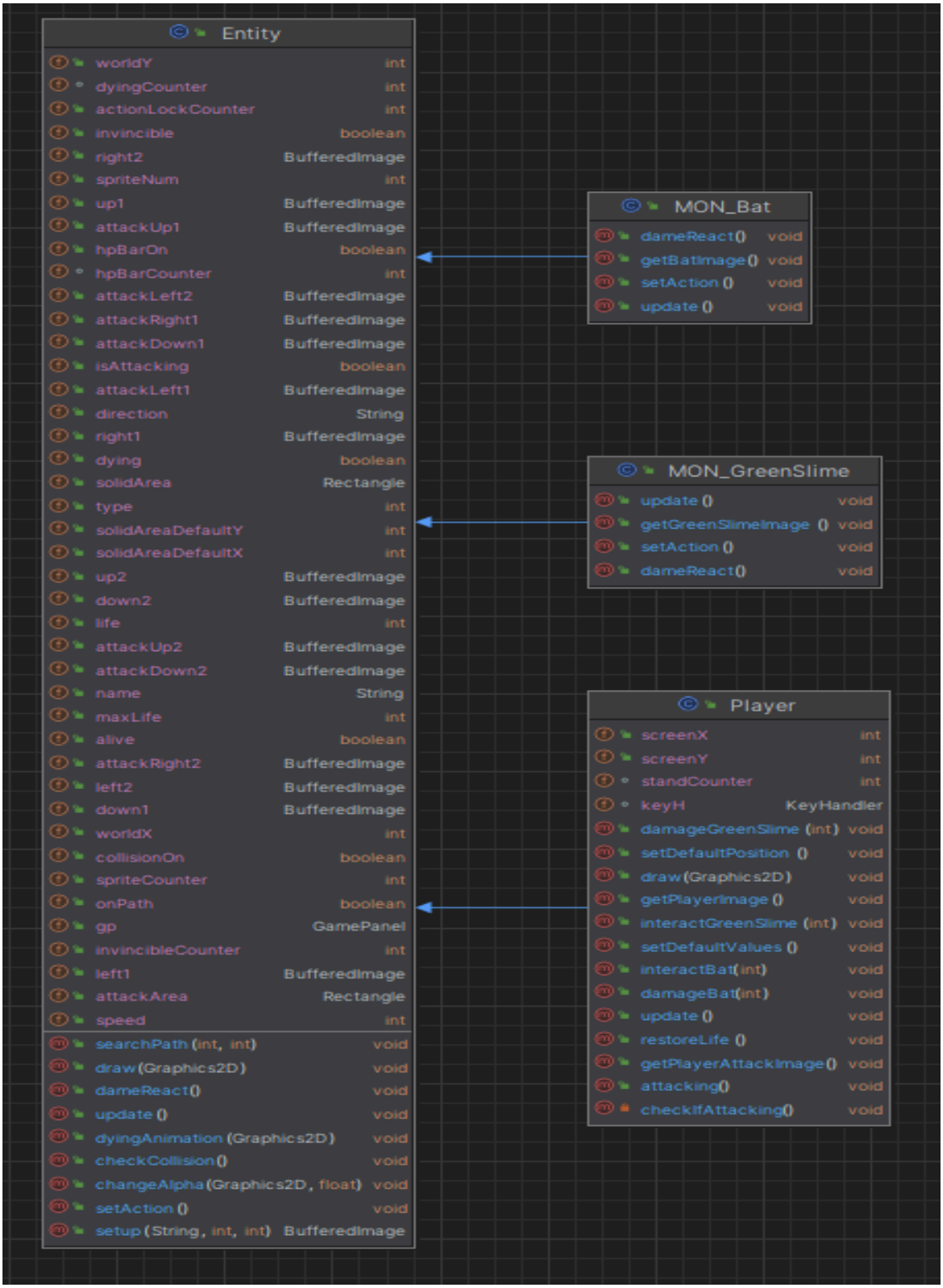
Last Time Adventure	PLAY	Play the game
		Pause the game
		Resume the game
		Quit the game
	OPTIONS	Change game setting (music,SE)
		Resume the game
		Quit the game

2.5 Use Case diagram

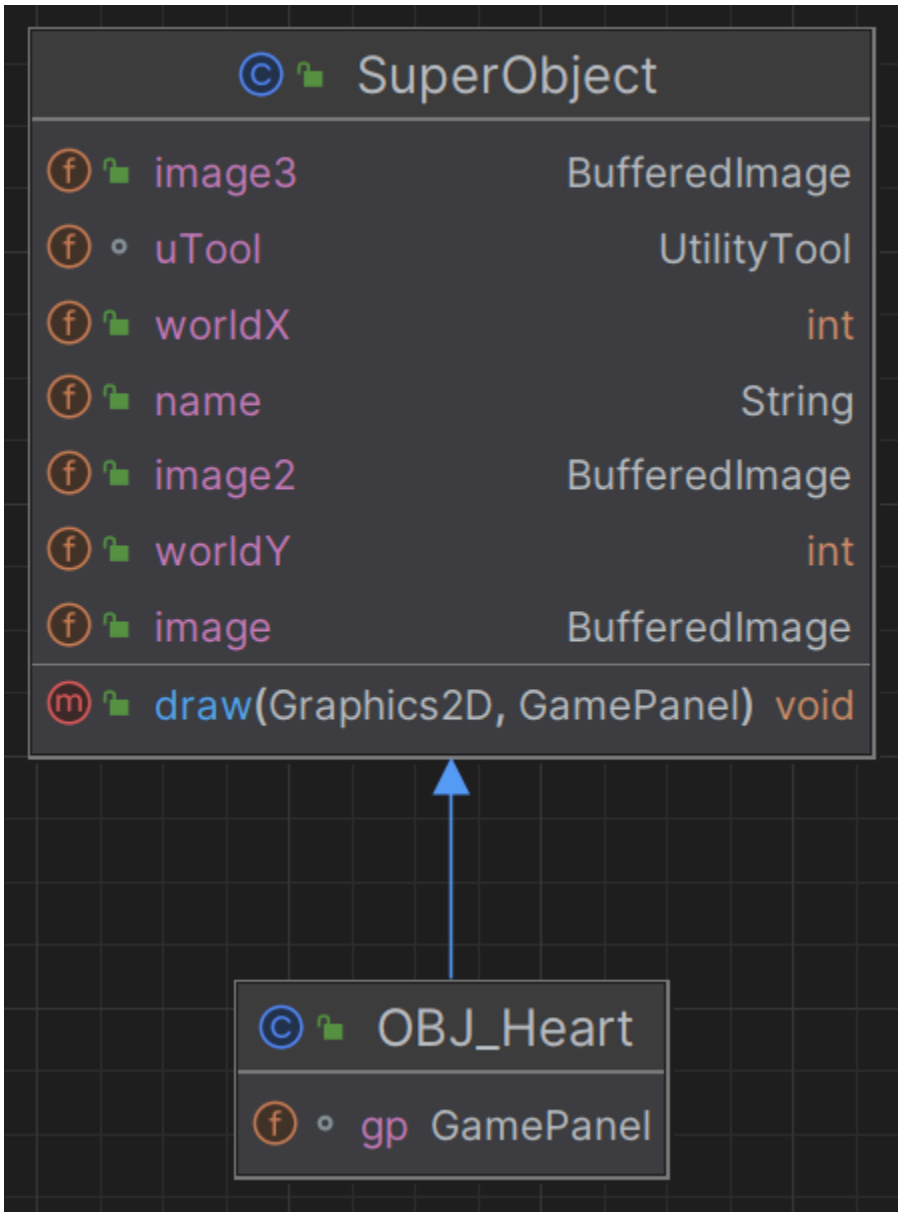


2.6 UML Diagram

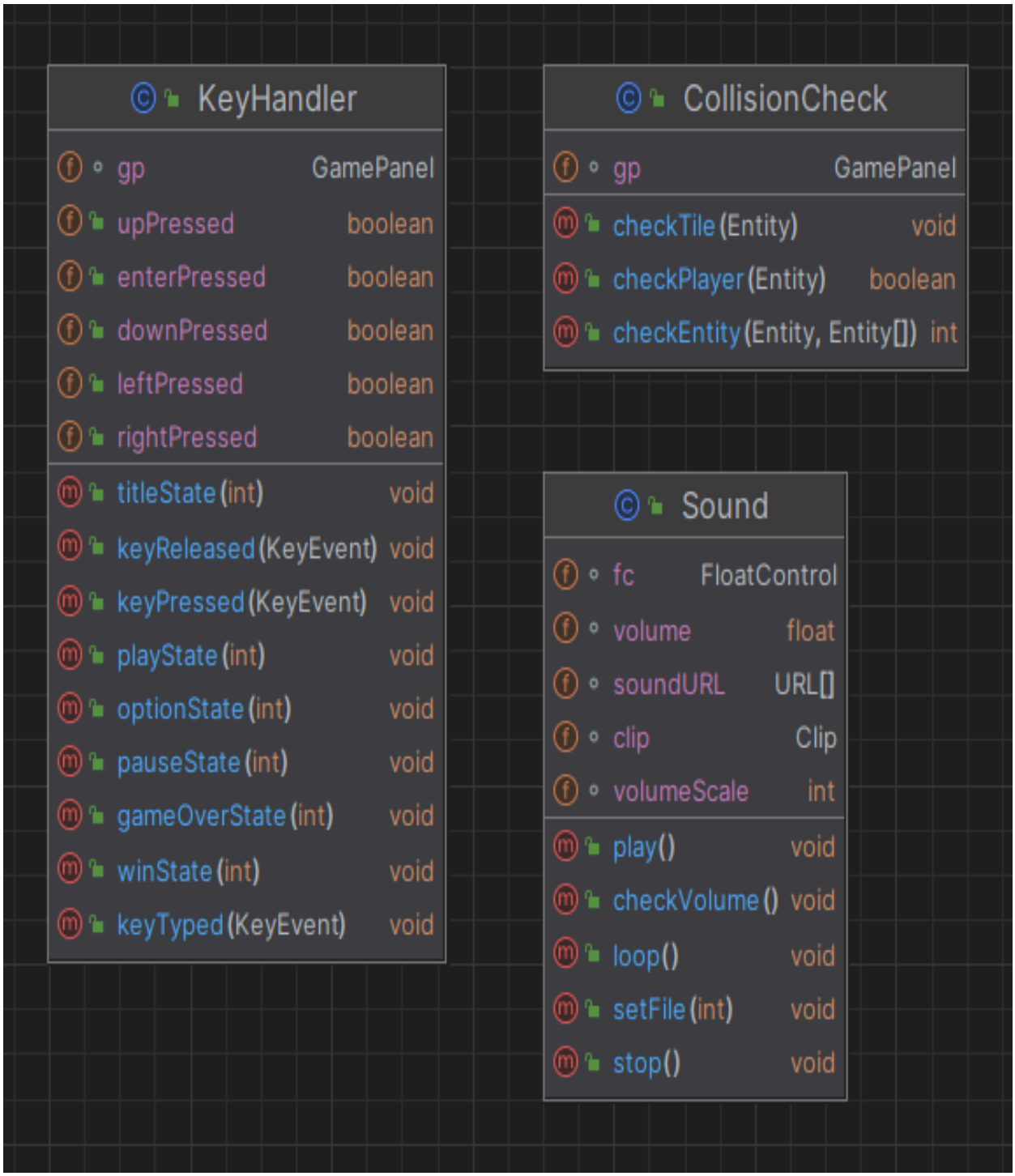
Entity Diagram



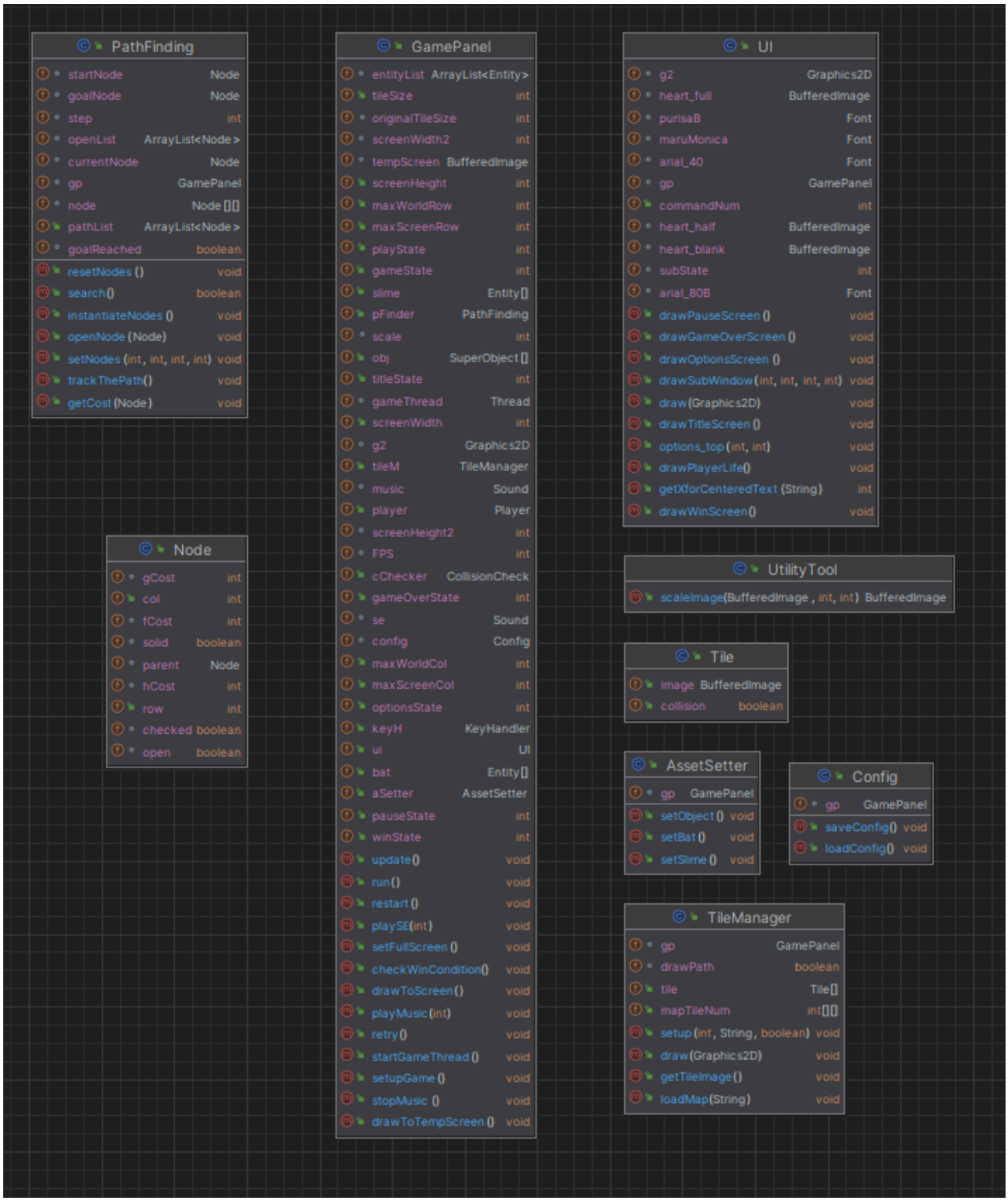
Object Diagram



Control Diagram



GUI Diagram


PathFinding
startNode Node
goalNode Node
step int
openList ArrayList<Node>
currentNode Node
gp GamePanel
node Node[][]
pathList ArrayList<Node>
goalReached boolean
resetNodes() void
search() boolean
instantiateNodes() void
openNode(Node) void
setNodes(int, int, int, int) void
trackThePath() void
getCost(Node) void
GamePanel
entityList ArrayList<Entity>
tileSize int
originalTileSize int
screenWidth2 int
tempScreen BufferedImage
screenHeight int
maxWorldRow int
maxScreenRow int
playState int
gameState int
slime Entity[]
pFinder PathFinding
scale int
obj SuperObject[]
titleState int
gameThread Thread
screenWidth int
g2 Graphics2D
tileM TileManager
music Sound
player Player
screenHeight2 int
FPS int
cChecker CollisionCheck
gameOverState int
se Sound
config Config
maxWorldCol int
maxScreenCol int
optionsState int
keyH KeyHandler
ui UI
bat Entity[]
aSetter AssetSetter
pauseState int
winState int
update() void
run() void
restart() void
playSE(int) void
setFullScreen() void
checkWinCondition() void
drawToScreen() void
playMusic(int) void
retry() void
startGameThread() void
setupGame() void
stopMusic() void
drawToTempScreen() void
UI
g2 Graphics2D
heart_full BufferedImage
purisaB Font
maruMonica Font
arial_40 Font
gp GamePanel
commandNum int
heart_half BufferedImage
heart_blank BufferedImage
subState int
arial_80B Font
drawPauseScreen() void
drawGameOverScreen() void
drawOptionsScreen() void
drawSubWindow(int, int, int, int) void
draw(Graphics2D) void
drawTitleScreen() void
options_top(int, int) void
drawPlayerLife() void
getXforCenteredText(String) int
drawWinScreen() void
Node
gCost int
col int
fCost int
solid boolean
parent Node
hCost int
row int
checked boolean
open boolean
UtilityTool
scaleImage(BufferedImage, int, int) BufferedImage
Tile
image BufferedImage
collision boolean
AssetSetter
gp GamePanel
setObject() void
setBat() void
setSlime() void
Config
gp GamePanel
saveConfig() void
loadConfig() void
TileManager
gp GamePanel
drawPath boolean
tile Tile[]
mapTileNum int[][]
setup(int, String, boolean) void
draw(Graphics2D) void
getTileImage() void
loadMap(String) void

CHAPTER 3: DESIGN & IMPLEMENTATION

3.1 UI Design

The UI class in "Last Time Adventure" is a crucial component responsible for rendering various user interface elements in the game. This includes the title screen, player life indicators, messages, and the pause screen. The class leverages the Java AWT library for graphical operations, providing a seamless and engaging visual experience for players.

The primary purpose of the UI class is to manage and render different visual components based on the game's state. It handles drawing the player's life, displaying messages, rendering the title and pause screens, and managing font styles and images used throughout the game.

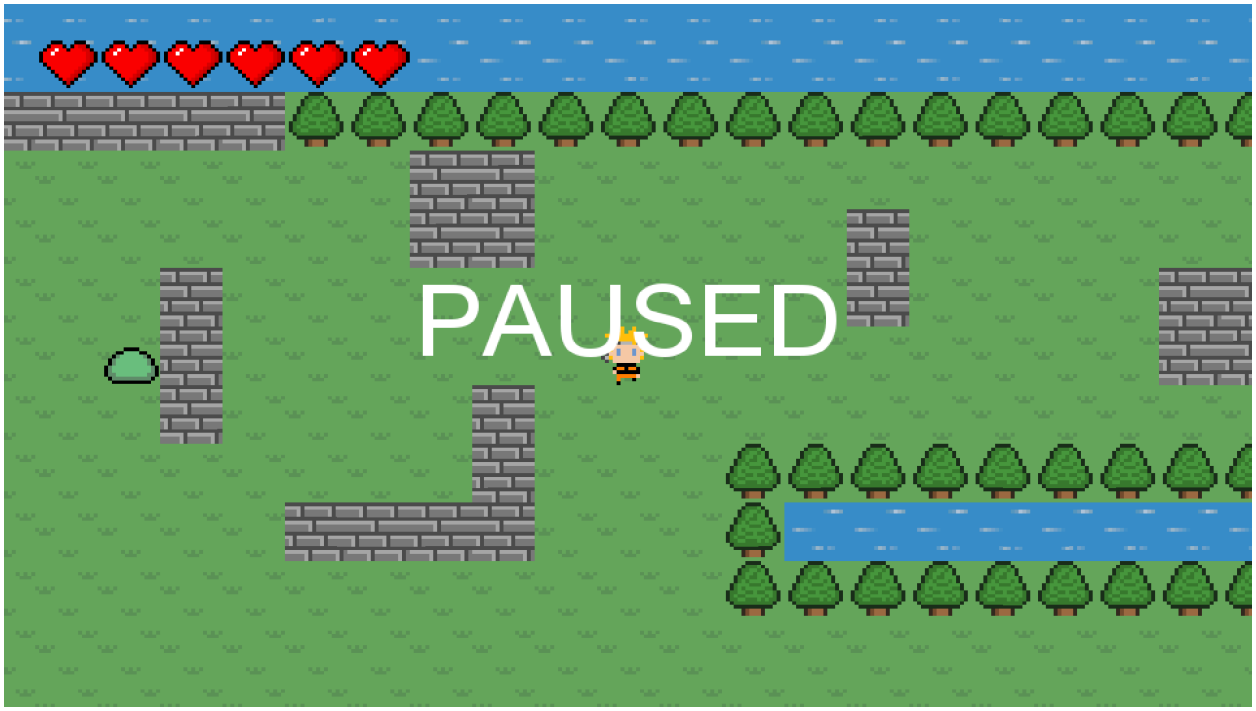
- UI class

Make title screen

```java
public void drawTitleScreen(){
    g2.setColor(new Color ( r: 0,  g: 0,  b: 0) ) ;
    g2.fillRect ( x: 0,  y: 0, gp. screenWidth, gp. screenHeight) ;

    //title name
    g2.setFont(g2.getFont().deriveFont(Font.BOLD,  size: 70F));
    String text = "Last Time Adventure";
    int x = getXforCenteredText(text);
    int y = gp.tileSize*3;

    //shadow
    g2.setColor(Color.GRAY);
    g2.drawString(text,  x: x+5,  y: y+5);
    //Main color
    g2.setColor(Color.WHITE);
    g2.drawString(text, x, y);

    //char image
    x = gp.screenWidth/2 - (gp.tileSize*2)/2;
    y += gp.tileSize*2;
    g2.drawImage(gp.player.down1,x,y, width: gp.tileSize*2, height: gp.tileSize*2, observer: null);

    //Menu
    g2.setFont (g2.getFont().deriveFont(Font.BOLD, size: 40F ));
    text ="NEW GAME";
    x = getXforCenteredText (text) ;
    y += gp.tileSize*4;
    g2.drawString (text, x, y) ;
    if(commandNum == 0){
        g2.drawString( str: ">", x: x-gp.tileSize,y);
    }

    text = "QUIT";
    x = getXforCenteredText (text) ;
    y+= gp.tileSize;
    g2.drawString (text, x, y);
    if(commandNum == 1){
        g2.drawString( str: ">", x: x-gp.tileSize,y);
    }
}
```

This function use to make title screen

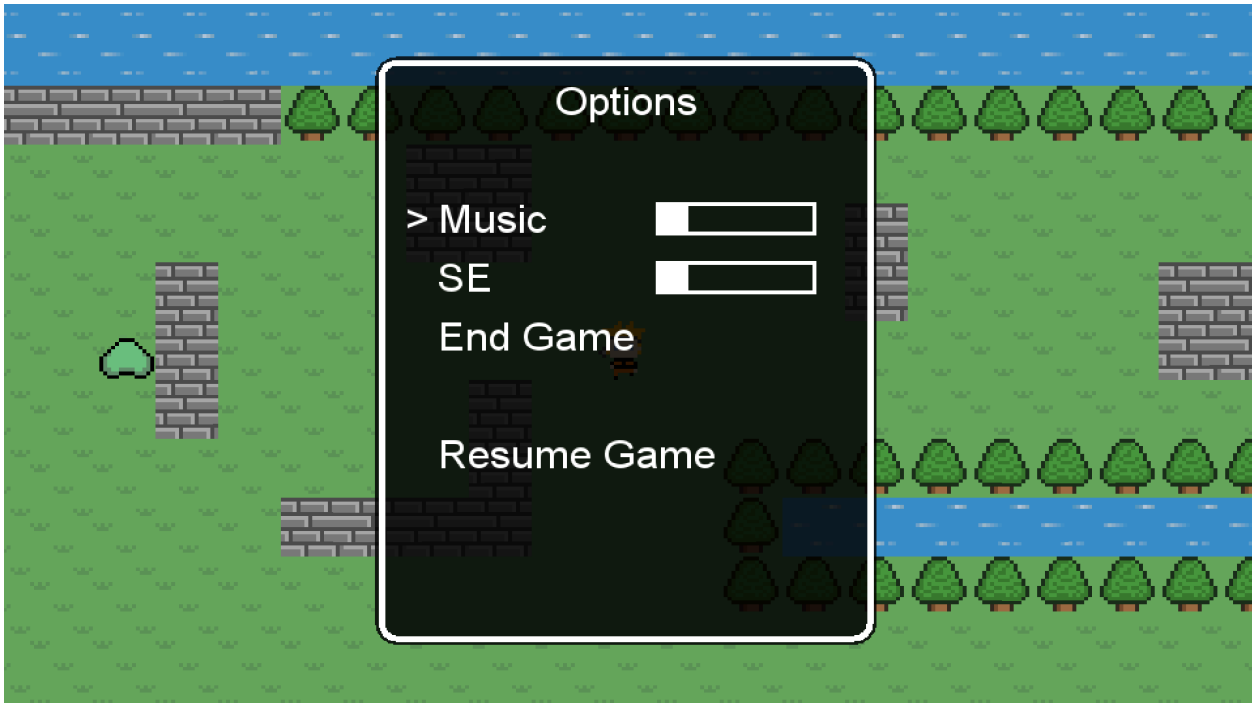
Make pause screen




```
public void drawPauseScreen(){
    g2.setFont(g2.getFont().deriveFont(Font.PLAIN, size: 80F));
    String text = "PAUSED";
    int x = getXforCenteredText(text);

    int y = gp.screenHeight/2;
    g2.drawString(text, x , y );
}
```

This function use to make pause screen

Make options screen




```
public void drawOptionsScreen(){
    g2.setColor(Color.white);
    g2.setFont(g2.getFont().deriveFont( size: 32f));

    //Sub window
    int frameX = gp.tileSize*6;
    int frameY = gp.tileSize;
    int frameWidth = gp.tileSize*8;
    int frameHeight = gp.tileSize*10;
    drawSubWindow(frameX,frameY,frameWidth,frameHeight);

    switch (subState){
        case 0: options_top(frameX, frameY); break;
        case 1:  break;
        case 2:  break;
    }
    gp.keyH.enterPressed = false;
}
```

```java
public void options_top(int frameX, int frameY){
    int textX;
    int textY;

    //Title
    String text = "Options";
    textX = getXforCenteredText(text);
    textY = frameY + gp.tileSize;
    g2.drawString(text,textX,textY);

    //Music
    textX = frameX + gp.tileSize;
    textY += gp.tileSize*2;
    //textY += gp.tileSize;
    g2.drawString( str: "Music", textX, textY);
    if(commandNum == 0){
        g2.drawString( str: ">", x: textX - 25,textY);
    }

    //SE
    textY += gp.tileSize;
    g2.drawString( str: "SE", textX, textY);
    if(commandNum == 1){
        g2.drawString( str: ">", x: textX - 25,textY);
    }

    //End Game
    textY += gp.tileSize;
    g2.drawString( str: "End Game", textX, textY);
    if(commandNum == 2){
        g2.drawString( str: ">", x: textX - 25,textY);
        if(gp.keyH.enterPressed == true){
            gp.playSE( i: 7);
            subState = 0;
            gp.gameState = gp.titleState;
            gp.stopMusic();
            gp.playMusic( i: 0);
            gp.restart();
        }
    }
```

```
    //Resume
    textY += gp.tileSize*2;
    g2.drawString( str: "Resume Game", textX, textY);
    if(commandNum == 3){
        g2.drawString( str: ">", x: textX - 25,textY);
        if(gp.keyH.enterPressed == true){
            gp.gameState = gp.playState;
            commandNum = 0;
        }
    }

    //Music Volume
    textX = frameX + (int)(gp.tileSize*4.5);
    textY = frameY + gp.tileSize*2 + 24;
    g2.setStroke(new BasicStroke( width: 3));
    g2.drawRect(textX,textY, width: 120, height: 24);
    int volumeWidth = 24 * gp.music.volumeScale;
    g2.fillRect(textX,textY,volumeWidth, height: 24);

    //SE Volume
    textY += gp.tileSize;
    g2.drawRect(textX,textY, width: 120, height: 24);
    volumeWidth = 24 * gp.se.volumeScale;
    g2.fillRect(textX,textY,volumeWidth, height: 24);

    gp.config.saveConfig();
}
```

This function use to make options screen

Make game over screen

```java
public void drawGameOverScreen(){

    g2.setColor(new Color( r: 0, g: 0, b: 0, a: 150));
    g2.fillRect( x: 0, y: 0,gp.screenWidth,gp.screenHeight);

    int x;
    int y;
    String text;
    g2.setFont(g2.getFont().deriveFont(Font.BOLD, size: 110f));

    text = "Game Over";
    //Shadow
    g2.setColor(Color.BLACK);
    x = getXforCenteredText(text);
    y = gp.tileSize*4;
    g2.drawString(text,x,y);
    //Main
    g2.setColor(Color.WHITE);
    g2.drawString(text, x: x - 4, y: y - 4);

    //Retry
    g2.setFont(g2.getFont().deriveFont( size: 50f));
    text = "Retry";
    x = getXforCenteredText(text);
    y += gp.tileSize*4;
    g2.drawString(text,x,y);
    if(commandNum == 0){
        g2.drawString( str: ">", x: x - 40,y);
    }

    //Back to title
    text = "Quit";
    x = getXforCenteredText(text);
    y += 55;
    g2.drawString(text,x,y);
    if(commandNum == 1){
        g2.drawString( str: ">", x: x - 40,y);
    }
}
```

This function use to make gameover screen

Make win game screen

```java
public void drawWinScreen(){
    g2.setColor(new Color( r: 0, g: 0, b: 0, a: 150));
    g2.fillRect( x: 0, y: 0,gp.screenWidth,gp.screenHeight);

    int x;
    int y;
    String text;
    g2.setFont(g2.getFont().deriveFont(Font.BOLD, size: 110f));

    text = "Winner!!";
    //Shadow
    g2.setColor(Color.BLACK);
    x = getXforCenteredText(text);
    y = gp.tileSize*4;
    g2.drawString(text,x,y);
    //Main
    g2.setColor(Color.WHITE);
    g2.drawString(text, x: x - 4, y: y - 4);

    //Back to title
    g2.setFont(g2.getFont().deriveFont( size: 50f));
    text = "Back To Title Screen";
    x = getXforCenteredText(text);
    y += gp.tileSize*4;
    g2.drawString(text,x,y);
    if(commandNum == 0){
        g2.drawString( str: ">", x: x - 40,y);
    }

}
```

This function use to make win screen

Do full screen

```
public void drawSubWindow(int x, int y, int width, int height) {
    Color c = new Color( r: 0,  g: 0,  b: 0,  a: 210);
    g2.setColor(c);
    g2.fillRoundRect(x, y, width, height,  arcWidth: 35,  arcHeight: 35);

    c = new Color( r: 255,  g: 255,  b: 255);
    g2.setColor(c);
    g2.setStroke(new BasicStroke( width: 5));
    g2.drawRoundRect( x: x + 5,  y: y + 5,  width: width - 10,  height: height - 10,  arcWidth: 25,  arcHeight: 25);
}
```

This function use to make full screen for game

- UtilityTool

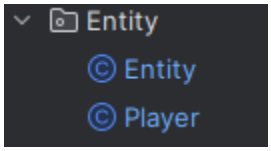
```
package Main;

import ...

9 usages   nghianguyen19 *
public class UtilityTool {
    5 usages   nghianguyen19
    public BufferedImage scaleImage(BufferedImage original, int width, int height ){
        BufferedImage scaledImage = new BufferedImage(width, height, original.getType());
        Graphics2D g2 = scaledImage.createGraphics();
        g2.drawImage(original,  x: 0,  y: 0 , width, height,  observer: null);
        g2.dispose();

        return scaledImage ;
    }
}
```

The UtilityTool class provides a utility method for scaling images. This method is useful for resizing images while preserving their aspect ratio. It's commonly used in graphical applications to adjust image dimensions based on requirements such as display size or layout constraints.

3.2 Entity , Object and Map Design

The Entity class serves as a blueprint for all in-game entities, including the player and various enemies. It handles essential aspects such as movement, state management, collision detection, and rendering. This class ensures that all entities behave consistently and interact properly within the game world.

The primary purpose of the Entity class is to encapsulate the properties and behaviors shared by all entities in the game. It provides methods for updating entity states, handling collisions, and rendering the entities on the screen. This unified approach allows for easy management and extension of entity behaviors.



- Entity

```
public void checkCollision(){
    collisionOn = false;
    gp.cChecker.checkTile( entity: this);
    gp.cChecker.checkEntity( entity: this, gp.slime);
    gp.cChecker.checkEntity( entity: this, gp.bat);
    boolean contactPlayer = gp.cChecker.checkPlayer( entity: this);

    if(this.type == 2 && contactPlayer){
        if(!gp.player.invincible){
            gp.playSE( i: 2);
            //the damage
            gp.player.life -= 1;
            gp.player.invincible = true;
        }
    }
}
```

This function is used to check for collisions between the game object and various other objects in the game (like tiles, slimes, bats, and the player).

- Player

```java
public void draw(Graphics2D g2){
    BufferedImage image = null;
    int screenX = worldX - gp.player.worldX + gp.player.screenX;
    int screenY = worldY - gp.player.worldY + gp.player.screenY;

    if(worldX + gp.tileSize > gp.player.worldX - gp.player.screenX &&
            worldX - gp.tileSize < gp.player.worldX + gp.player.screenX &&
            worldY + gp.tileSize > gp.player.worldY - gp.player.screenY &&
            worldY - gp.tileSize < gp.player.worldY + gp.player.screenY){

        switch (direction) {
            case "up":
                if(spriteNum == 1){image = up1;}
                if (spriteNum == 2){image = up2;}
                break;
            case "down":
                if(spriteNum == 1){image = down1;}
                if (spriteNum == 2){image = down2;}
                break;
            case "left":
                if(spriteNum == 1){image = left1;}
                if (spriteNum == 2){image = left2;}
                break;
            case "right":
                if(spriteNum == 1){image = right1;}
                if (spriteNum == 2){image = right2;}
                break;
        }
        //Monster healthBar
        if(type == 1 && hpBarOn == true){
            double oneScale = (double)gp.tileSize/maxLife;
            double hpBarValue = oneScale * life;

            //outline
            g2.setColor(new Color( r: 35, g: 35, b: 35));
            g2.fillRect( x: screenX - 1, y: screenY - 3, width: gp.tileSize + 2, height: 7);
            //fill color
            g2.setColor(new Color( r: 255, g: 0, b: 30));
            g2.fillRect(screenX, y: screenY - 2, (int) hpBarValue, height: 5);
```

```
            hpBarCounter++;
            if(hpBarCounter > 600){
                hpBarCounter = 0;
                hpBarOn = false;
            }
        }
        //Start Alpha Monster
        if(invincible == true){
            hpBarOn = true;
            hpBarCounter = 0;
            changeAlpha(g2, alphaValue: 0.4f);
        }
        if(dying == true){
            dyingAnimation(g2);
        }
        g2.drawImage(image, screenX, screenY, gp.tileSize, gp.tileSize, observer: null);
        //Reset alpha Monster
        changeAlpha(g2, alphaValue: 1f);
    }
}
```

The draw method is responsible for rendering a game entity like character and monsters on the screen using Graphics2D. It determines whether the entity is within the visible area of the screen and draws it with the appropriate sprite and optional effects such as health bars, invincibility transparency, and a dying animation.

```
public void update(){
    setAction();
    checkCollision();
    // If collision is false, player can move
    if(!collisionOn){
        switch (direction){
            case "up":
                worldY -= speed;
                break;
            case "down":
                worldY += speed;
                break;
            case "left":
                worldX -= speed;
                break;
            case "right":
                worldX += speed;
                break;
        }
    }
    spriteCounter++;
    if(spriteCounter > 20){
        if(spriteNum == 1){
            spriteNum = 2;
        } else if (spriteNum == 2) {
            spriteNum = 1;
        }
        spriteCounter = 0;
    }

    if(invincible){
        invincibleCounter++;
        if(invincibleCounter > 40){
            invincible = false;
            invincibleCounter = 0;
        }
    }
}
```

The update method is responsible for updating the state of the game entity in each frame of the game loop. It handles movement, sprite animation, and invincibility duration.

```
public void dyingAnimation(Graphics2D g2){
    dyingCounter++;
    int i = 5;
    if(dyingCounter <= i){changeAlpha(g2, alphaValue: 0f);}
    if(dyingCounter > i && dyingCounter <= i*2){changeAlpha(g2, alphaValue: 1f);}
    if(dyingCounter > i*2 && dyingCounter <= i*3){changeAlpha(g2, alphaValue: 0f);}
    if(dyingCounter > i*3 && dyingCounter <= i*4){changeAlpha(g2, alphaValue: 1f);}
    if(dyingCounter > i*4 && dyingCounter <= i*5){changeAlpha(g2, alphaValue: 0f);}
    if(dyingCounter > i*5 && dyingCounter <= i*6){changeAlpha(g2, alphaValue: 1f);}
    if(dyingCounter > i*6 && dyingCounter <= i*7){changeAlpha(g2, alphaValue: 0f);}
    if(dyingCounter > i*7 && dyingCounter <= i*8){changeAlpha(g2, alphaValue: 1f);}
    if(dyingCounter > i*8){
        dying = false;
        alive = false;
    }
}
```

The ‘dyingAnimation’ method manages the visual fading effect of an entity when it is dying. It achieves this by gradually changing the alpha transparency of the entity's graphics over time.

```
public BufferedImage setup (String imagePath, int width, int height){
    UtilityTool uTool = new UtilityTool();
    BufferedImage image = null;
    try {
        image = ImageIO.read(getClass().getResourceAsStream( name: imagePath + ".png"));
        image = uTool.scaleImage(image, width, height);
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
    return image;
}
```

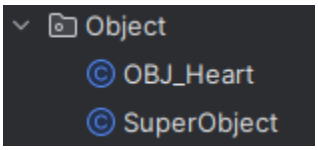
The ‘setup’ method is designed to load an image from a given path, scale it to the specified dimensions, and return the resulting BufferedImage. This method leverages the UtilityTool class for scaling the image.

```java
public void searchPath(int goalCol, int goalRow) {
    int startCol = (worldX + solidArea.x) / gp.tileSize;
    int startRow = (worldY + solidArea.y) / gp.tileSize;
    gp.pFinder.setNodes(startCol, startRow, goalCol, goalRow);
    if (gp.pFinder.search() == true) {
        int nextX = gp.pFinder.pathList.get(0).col * gp.tileSize;
        int nextY = gp.pFinder.pathList.get(0).row * gp.tileSize;

        //Entity solidArea position
        int enLeftX = worldX + solidArea.x;
        int enRightX = worldX + solidArea.x + solidArea.width;
        int enTopY = worldY + solidArea.y;
        int enBottomY = worldY + solidArea.y + solidArea.height;
        if (enTopY > nextY && enLeftX >= nextX && enRightX < nextX +gp.tileSize){
            direction = "up";
        } else if (enTopY < nextY && enLeftX >= nextX && enRightX < nextX +gp.tileSize) {
            direction = "down";
        } else if (enTopY >= nextY && enBottomY < nextY  + gp.tileSize) {
            //Left or right
            if (enLeftX > nextX){
                direction = "left";
            }
            if(enLeftX < nextX){
                direction = "right";
            }
        } else if (enTopY > nextY && enLeftX > nextX) {
            //up or left
            direction = "up";
            checkCollision();
            if(collisionOn == true){
                direction = "left";
            }
        } else if (enTopY > nextY && enLeftX < nextX) {
            //up or right
            direction = "up";
            checkCollision();
            if(collisionOn == true){
                direction = "right";
            }
```

```
        } else if (enTopY < nextY && enLeftX > nextX) {
            //down or left
            direction = "down";
            checkCollision();
            if (collisionOn == true) {
                direction = "left";
            }
        } else if (enTopY < nextY && enLeftX < nextX) {
            //down or right
            direction = "down";
            checkCollision();
            if(collisionOn == true){
                direction = "right";
            }
        }
    }
}
```

The searchPath method is designed to find a path for an entity from its current position to a specified goal position (goalCol, goalRow). It uses a pathfinding algorithm implemented in the PathFinder class (gp.pFinder) to determine the next steps to take.

- OBJ_Heart




```java
package Object;

import ...


2 usages  nghianguyen19 +1
public class OBJ_Heart extends SuperObject  {
    1 usage
    GamePanel gp;

    1 usage  nghianguyen19 +1
    public OBJ_Heart (GamePanel gp){
        this.gp =gp ;

        name = "Heart";
        try{
            image = ImageIO.read(getClass().getResourceAsStream( name: "/Picture/object/heart_full.png"));
            image2 = ImageIO.read(getClass().getResourceAsStream( name: "/Picture/object/heart_half.png"));
            image3 = ImageIO.read(getClass().getResourceAsStream( name: "/Picture/object/heart_blank.png"));

            image =  uTool.scaleImage(image, gp.tileSize, gp.tileSize);
            image2 = uTool.scaleImage(image2, gp.tileSize, gp.tileSize);
            image3 = uTool.scaleImage(image3, gp.tileSize, gp.tileSize);
        }catch (Exception e) {
            throw new RuntimeException(e);
        }

    }

}
```

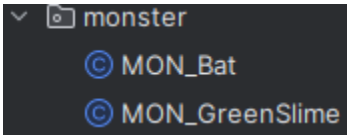
The OBJ_Heart class represents the heart object, used to display the player's health status. This class extends SuperObject. The OBJ_Heart class initializes three different heart images (full, half, and empty) and scales them to fit the game's tile size.

- SuperObject

```java
package Object;
import ...

6 usages  1 inheritor  nghianguyen19 +1
public class SuperObject {
    5 usages
    public BufferedImage image, image2, image3;
    1 usage
    public String name;
    3 usages
    public int worldX, worldY;

    3 usages
    UtilityTool uTool = new UtilityTool();

    1 usage  nghianguyen19
    public void draw(Graphics2D g2, GamePanel gp){
        int screenX = worldX - gp.player.worldX + gp.player.screenX;
        int screenY = worldY - gp.player.worldY + gp.player.screenY;

        if(worldX + gp.tileSize > gp.player.worldX - gp.player.screenX &&
            worldX - gp.tileSize < gp.player.worldX + gp.player.screenX &&
            worldY + gp.tileSize > gp.player.worldY - gp.player.screenY &&
            worldY - gp.tileSize < gp.player.worldY + gp.player.screenY){
            g2.drawImage(image, screenX, screenY, gp.tileSize, gp.tileSize,  observer: null);
        }
    }

}
```

The SuperObject class represents an object within the game world. This class includes methods and attributes for managing and drawing the object.

- MON_Bat




```
public class MON_Bat extends Entity {
    2 usages  nguyeenphan +1
    public MON_Bat(GamePanel gp) {
        super(gp);
        type = 1;
        direction = "down";
        name = "Bat";
        speed = 3;
        maxLife = 2;
        life = maxLife;

        solidArea.x = 3;
        solidArea.y = 18;
        solidArea.width = 42;
        solidArea.height = 30;
        solidAreaDefaultX = solidArea.x;
        solidAreaDefaultY = solidArea.y;

        getBatImage();
    }
```

```
public void getBatImage(){
        up1 = setup( imagePath: "/Picture/bat/bat_down_1", gp.tileSize, gp.tileSize);
        up2 = setup( imagePath: "/Picture/bat/bat_down_2", gp.tileSize, gp.tileSize);
        down1 = setup ( imagePath: "/Picture/bat/bat_down_1", gp.tileSize, gp.tileSize);;
        down2 = setup( imagePath: "/Picture/bat/bat_down_2", gp.tileSize, gp.tileSize);
        left1 = setup( imagePath: "/Picture/bat/bat_down_1", gp.tileSize, gp.tileSize);
        left2 = setup( imagePath: "/Picture/bat/bat_down_2", gp.tileSize, gp.tileSize);
        right1 = setup( imagePath: "/Picture/bat/bat_down_1", gp.tileSize, gp.tileSize);
        right2 = setup( imagePath: "/Picture/bat/bat_down_2", gp.tileSize, gp.tileSize);
}
```

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

    int xDistance = Math.abs(worldX - gp.player.worldX);
    int yDistance = Math.abs(worldY - gp.player.worldY);
    int tileDistance = (xDistance + yDistance)/gp.tileSize;
    if(onPath == false && tileDistance < 5){
        int i = new Random().nextInt( bound: 100)+1;
        if(i > 50){
            onPath = true;
        }
    }
}
```

```java
@Override
public void setAction(){
    if(onPath == true){
        int goalCol = (gp.player.worldX + gp.player.solidArea.x)/gp.tileSize;
        int goalRow = (gp.player.worldY + gp.player.solidArea.y)/gp.tileSize;
        searchPath(goalCol,goalRow);
    }
    else{
        actionLockCounter++;
        if (actionLockCounter == 120) {
            Random random = new Random();
            int i = random.nextInt( bound: 100) + 1; // pick up a number from 1 to 100
            if (i <= 25) {
                direction = "up";
            }
            if (i > 25 && i <= 50) {
                direction = "down";
            }
            if (i > 50 && i <= 75) {
                direction = "left";
            }
            if (i > 75) {
                direction = "right";
            }
            actionLockCounter = 0;
        }
    }
}
2 usages  Thanh Nguyễn +1
@Override
public void dameReact(){
    actionLockCounter = 0;
    onPath = true;
}
```

The MON_Bat class represents a bat monster entity. It extends the Entity class and includes various attributes and methods for managing the bat's behavior, appearance, and interactions within the game world.

- MON_GreenSlime

```
public MON_GreenSlime(GamePanel gp) {
    super(gp);
    type = 1;
    name = "Green Slime";
    direction = "down";
    speed = 1;
    maxLife = 4;
    life = maxLife;

    solidArea.x = 3;
    solidArea.y = 18;
    solidArea.width = 42;
    solidArea.height = 30;
    solidAreaDefaultX = solidArea.x;
    solidAreaDefaultY = solidArea.y;

    getGreenSlimeImage();
}
1 usage  Thanh Nguyễn
public void getGreenSlimeImage(){
    up1 = setup( imagePath: "/Picture/green_slime/green_slime_down_1", gp.tileSize, gp.tileSize);
    up2 = setup( imagePath: "/Picture/green_slime/green_slime_down_2", gp.tileSize, gp.tileSize);
    down1 = setup ( imagePath: "/Picture/green_slime/green_slime_down_1", gp.tileSize, gp.tileSize);
    down2 = setup( imagePath: "/Picture/green_slime/green_slime_down_2", gp.tileSize, gp.tileSize);
    left1 = setup( imagePath: "/Picture/green_slime/green_slime_down_1", gp.tileSize, gp.tileSize);
    left2 = setup( imagePath: "/Picture/green_slime/green_slime_down_2", gp.tileSize, gp.tileSize);
    right1 = setup( imagePath: "/Picture/green_slime/green_slime_down_1", gp.tileSize, gp.tileSize);
    right2 = setup( imagePath: "/Picture/green_slime/green_slime_down_2", gp.tileSize, gp.tileSize);
}
```

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

    int xDistance = Math.abs(worldX - gp.player.worldX);
    int yDistance = Math.abs(worldY - gp.player.worldY);
    int tileDistance = (xDistance + yDistance)/gp.tileSize;
    if(onPath == false && tileDistance < 5){
        int i = new Random().nextInt( bound: 100)+1;
        if(i > 50){
            onPath = true;
        }
    }
}

1 usage  nguyeenphan +2
@Override
public void setAction(){
    if(onPath == true){
        int goalCol = (gp.player.worldX + gp.player.solidArea.x)/gp.tileSize;
        int goalRow = (gp.player.worldY + gp.player.solidArea.y)/gp.tileSize;
        searchPath(goalCol,goalRow);
    }
    else{
        actionLockCounter++;
        if (actionLockCounter == 120) {
            Random random = new Random();
            int i = random.nextInt( bound: 100) + 1; // pick up a number from 1 to 100
            if (i <= 25) {
                direction = "up";
            }
            if (i > 25 && i <= 50) {
                direction = "down";
            }
            if (i > 50 && i <= 75) {
                direction = "left";
            }
            if (i > 75) {
                direction = "right";
            }
            actionLockCounter = 0;
        }
    }
}
```

```java
@Override
public void dameReact(){
    actionLockCounter = 0;
    onPath = true;
}
```

The MON_GreenSlime class represents a slime monster entity. It extends the Entity class and includes various attributes and methods for managing the bat's behavior, appearance, and interactions within the game world.

- Tile: Set value for the TileManager

```
public class Tile {
    4 usages
    public BufferedImage image;
    10 usages
    public boolean collision = false;
}
```

- TileManager: The TileManager class is responsible for managing the tiles, including loading tile images, setting up the map, and drawing the tiles on the screen.

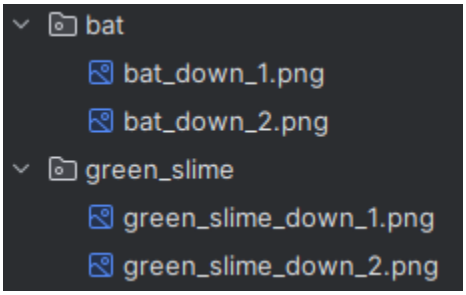
```java
public class TileManager {
    30 usages
    GamePanel gp;
    16 usages
    public Tile[] tile;
    12 usages
    public int[][] mapTileNum;
    no usages
    boolean drawPath = true;
    1 usage  Thanh Nguyên
    public TileManager(GamePanel gp){
        this.gp = gp;
        tile = new Tile[10];
        mapTileNum = new int[gp.maxWorldCol][gp.maxWorldRow];
        getTileImage();
        loadMap( filePath: "/Picture/map/worldM.txt");
    }
    1 usage  nghianguyen19 +1
    public void getTileImage(){
            setup( index: 0,  imagePath: "grass",  collision: false);
            setup( index: 1,  imagePath: "wall",  collision: true);
            setup( index: 2,  imagePath: "water",  collision: true);
            setup( index: 3,  imagePath: "tree",  collision: true);
            setup( index: 4,  imagePath: "dirt", collision: false );
            setup( index: 5,  imagePath: "sand",  collision: false);
     }

    6 usages  nghianguyen19 +1
    public void setup(int index, String imagePath, boolean collision ){

        UtilityTool uTool = new UtilityTool();
        try{
            tile[index] = new Tile();
            tile[index].image = ImageIO.read(Objects.requireNonNull(getClass().getResourceAsStream( name: "/Picture/tiles/"+ imagePath+ ".png")));
            tile[index].image = uTool.scaleImage(tile[index].image, gp.tileSize, gp.tileSize);
            tile[index].collision =collision;

        }catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
```

```java
public void loadMap(String filePath){
    try{
        InputStream is = getClass().getResourceAsStream(filePath);
        BufferedReader br = new BufferedReader(new InputStreamReader(Objects.requireNonNull(is)));
        int col = 0;
        int row = 0;
        while(col < gp.maxWorldCol && row < gp.maxWorldRow) {
            String line = br.readLine();
            while (col < gp.maxWorldCol) {
                String[] numbers = line.split( regex: " ");
                int num = Integer.parseInt(numbers[col]);
                mapTileNum[col][row] = num;
                col++;
            }
            if (col == gp.maxWorldCol) {
                col = 0;
                row++;
            }
        }
        br.close();
    }catch (Exception e) {
        throw new RuntimeException(e);
    }
}
```

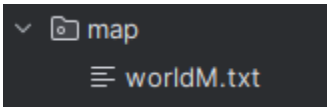
```java
public void draw(Graphics2D g2){
    int worldCol = 0;
    int worldRow = 0;

    while(worldCol < gp.maxWorldCol && worldRow < gp.maxWorldRow){
        int tileNum = mapTileNum[worldCol][worldRow];
        int worldX = worldCol * gp.tileSize;
        int worldY = worldRow * gp.tileSize;
        int screenX = worldX - gp.player.worldX + gp.player.screenX;
        int screenY = worldY - gp.player.worldY + gp.player.screenY;

        if(worldX + gp.tileSize > gp.player.worldX - gp.player.screenX &&
            worldX - gp.tileSize < gp.player.worldX + gp.player.screenX &&
            worldY + gp.tileSize > gp.player.worldY - gp.player.screenY &&
            worldY - gp.tileSize < gp.player.worldY + gp.player.screenY){
            g2.drawImage(tile[tileNum].image, screenX, screenY, gp.tileSize, gp.tileSize,  observer: null);
        }

        worldCol++;

        if(worldCol == gp.maxWorldCol){
            worldCol = 0;
            worldRow++;
        }
    }
```

This is the package that contain all character and monsters design in the game:

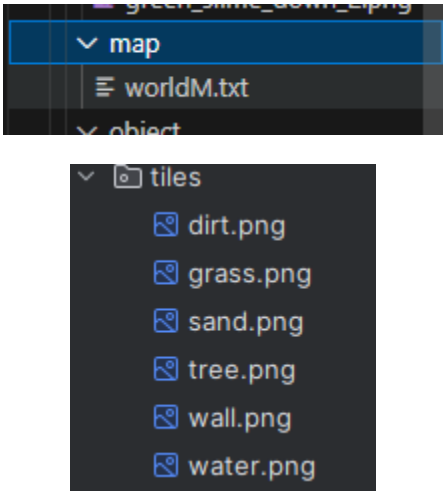
- Bat and green_slime : images for monsters bat and slime



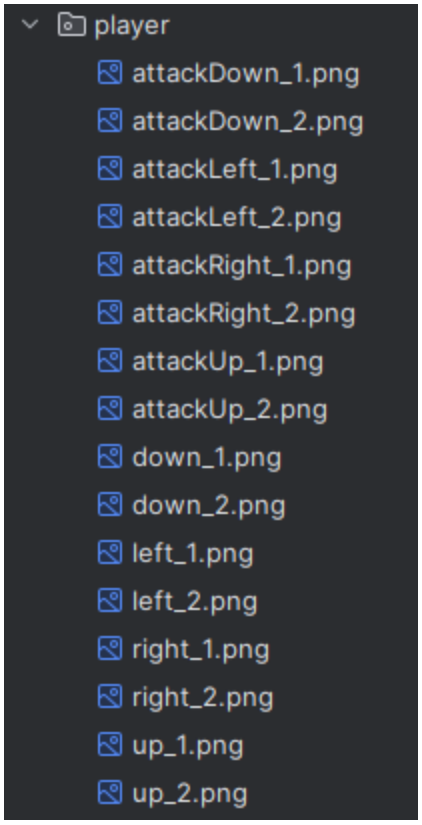
- Map: txt file use for draw map



- Tiles: The cells that make up the map

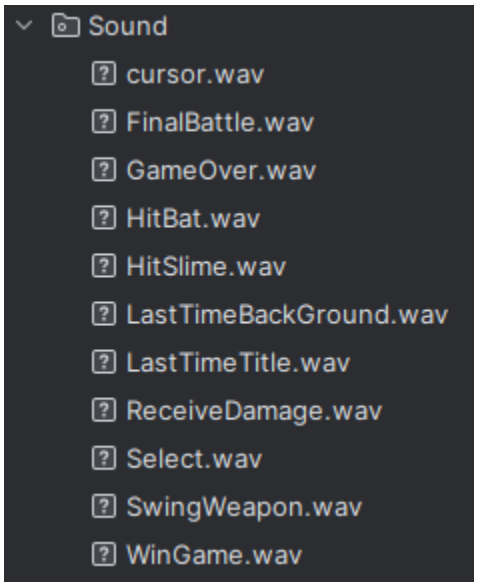


- Player and objects: Image of character movement and character health bar

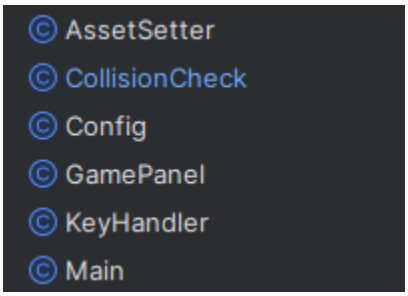


3.3 Sound Design

This is the package that contain all sound and sound effect in the game:



3.4 Game Mechanics Design



- AssetSetter: The AssetSetter class is responsible for initializing and placing various game entities, such as bats and slimes, at specific locations on the game map. This class uses a reference to the GamePanel to access and modify the game state.

```
package Main;

import ...
2 usages  Thanh Nguyễn +2
public class AssetSetter {
    31 usages
    GamePanel gp;

    1 usage  nghianguyen19
    public AssetSetter(GamePanel gp) { this.gp = gp ; }
    1 usage  Thanh Nguyễn
    public void setObject(){}
    3 usages  Thanh Nguyễn +1
    public void setBat(){
        gp.bat[0] = new MON_Bat(gp);
        gp.bat[0].worldX = 31 * gp.tileSize;
        gp.bat[0].worldY = 9 * gp.tileSize;

        gp.bat[1] = new MON_Bat(gp);
        gp.bat[1].worldX = 14 * gp.tileSize;
        gp.bat[1].worldY = 23 * gp.tileSize;

    }
    3 usages  Thanh Nguyễn
    public void setSlime(){
        gp.slime[0] = new MON_GreenSlime(gp);
        gp.slime[0].worldX = 14 * gp.tileSize;
        gp.slime[0].worldY = 9 * gp.tileSize;

        gp.slime[1] = new MON_GreenSlime(gp);
        gp.slime[1].worldX = 31 * gp.tileSize;
        gp.slime[1].worldY = 16 * gp.tileSize;

        gp.slime[2] = new MON_GreenSlime(gp);
        gp.slime[2].worldX = 31 * gp.tileSize;
        gp.slime[2].worldY = 23 * gp.tileSize;
    }
}
```

- CollisionCheck: The CollisionCheck class is designed to manage collision detection. It checks for collisions between entities and tiles, between entities and other entities, and between entities and the player.

```
public void checkTile(Entity entity) {
    int entityLeftWorldX = entity.worldX + entity.solidArea.x;
    int entityRightWorldX = entity.worldX + entity.solidArea.x + entity.solidArea.width;
    int entityTopWorldY = entity.worldY + entity.solidArea.y;
    int entityBottomWorldY = entity.worldY + entity.solidArea.y + entity.solidArea.height;

    int entityLeftCol = entityLeftWorldX / gp.tileSize;
    int entityRightCol = entityRightWorldX / gp.tileSize;
    int entityTopRow = entityTopWorldY / gp.tileSize;
    int entityBottomRow = entityBottomWorldY / gp.tileSize;
    int tileNum1, tileNum2;

    switch (entity.direction) {
        case "up":
            entityTopRow = (entityTopWorldY - entity.speed) / gp.tileSize;
            tileNum1 = gp.tileM.mapTileNum[entityLeftCol][entityTopRow];
            tileNum2 = gp.tileM.mapTileNum[entityRightCol][entityTopRow];
            if (gp.tileM.tile[tileNum1].collision || gp.tileM.tile[tileNum2].collision) {
                entity.collisionOn = true;
            }
            break;
        case "down":
            entityBottomRow = (entityBottomWorldY + entity.speed) / gp.tileSize;
            tileNum1 = gp.tileM.mapTileNum[entityLeftCol][entityBottomRow];
            tileNum2 = gp.tileM.mapTileNum[entityRightCol][entityBottomRow];
            if (gp.tileM.tile[tileNum1].collision || gp.tileM.tile[tileNum2].collision) {
                entity.collisionOn = true;
            }
            break;
        case "left":
            entityLeftCol = (entityLeftWorldX - entity.speed) / gp.tileSize;
            tileNum1 = gp.tileM.mapTileNum[entityLeftCol][entityTopRow];
            tileNum2 = gp.tileM.mapTileNum[entityLeftCol][entityBottomRow];
            if (gp.tileM.tile[tileNum1].collision || gp.tileM.tile[tileNum2].collision) {
                entity.collisionOn = true;
            }
            break;
```

```
        case "right":
            entityRightCol = (entityRightWorldX + entity.speed) / gp.tileSize;
            tileNum1 = gp.tileM.mapTileNum[entityRightCol][entityTopRow];
            tileNum2 = gp.tileM.mapTileNum[entityRightCol][entityBottomRow];
            if (gp.tileM.tile[tileNum1].collision || gp.tileM.tile[tileNum2].collision) {
                entity.collisionOn = true;
            }
            break;
    }
}
```

```
public int checkEntity(Entity entity, Entity[] target) {
    int index = 999;
    for (int i = 0; i < target.length; i++) {

        if (target[i] != null) {
            //get entity's solid area position
            entity.solidArea.x = entity.worldX + entity.solidArea.x;
            entity.solidArea.y = entity.worldY + entity.solidArea.y;
            //get the object's solid area position
            target[i].solidArea.x = target[i].worldX + target[i].solidArea.x;
            target[i].solidArea.y = target[i].worldY + target[i].solidArea.y;

            switch (entity.direction) {
                case "up":
                    entity.solidArea.y -= entity.speed;
                    break;
                case "down":
                    entity.solidArea.y += entity.speed;
                    break;
                case "left":
                    entity.solidArea.x -= entity.speed;
                    break;
                case "right":
                    entity.solidArea.x += entity.speed;
                    break;
            }
            if (entity.solidArea.intersects(target[i].solidArea)) {
                if (target[i] != entity) {
                    entity.collisionOn = true;
                    index = i;
                }
            }
            entity.solidArea.x = entity.solidAreaDefaultX;
            entity.solidArea.y = entity.solidAreaDefaultY;
            target[i].solidArea.x = target[i].solidAreaDefaultX;
            target[i].solidArea.y = target[i].solidAreaDefaultY;
        }
    }
    return index;
}
```

```java
public boolean checkPlayer(Entity entity) {
    boolean contactPlayer = false;
    //get entity's solid area position
    entity.solidArea.x = entity.worldX + entity.solidArea.x;
    entity.solidArea.y = entity.worldY + entity.solidArea.y;
    //get the object's solid area position
    gp.player.solidArea.x = gp.player.worldX + gp.player.solidArea.x;
    gp.player.solidArea.y = gp.player.worldY + gp.player.solidArea.y;

    switch (entity.direction) {
        case "up":
            entity.solidArea.y -= entity.speed;
            break;
        case "down":
            entity.solidArea.y += entity.speed;
            break;
        case "left":
            entity.solidArea.x -= entity.speed;
            break;
        case "right":
            entity.solidArea.x += entity.speed;

            break;
    }
    if (entity.solidArea.intersects(gp.player.solidArea)) {
        entity.collisionOn = true;
        contactPlayer = true;
    }
    entity.solidArea.x = entity.solidAreaDefaultX;
    entity.solidArea.y = entity.solidAreaDefaultY;
    gp.player.solidArea.x = gp.player.solidAreaDefaultX;
    gp.player.solidArea.y = gp.player.solidAreaDefaultY;
    return contactPlayer;
}
```

- Config: The Config class is responsible for saving and loading configuration settings for the game. Specifically, it handles the volume settings for music and sound effects (SE).

```
public class Config {
    5 usages
    GamePanel gp;
    1 usage  Thanh Nguyễn
    public Config(GamePanel gp) { this.gp = gp; }
    1 usage  Thanh Nguyễn
    public void saveConfig(){
        try {
            BufferedWriter bw = new BufferedWriter(new FileWriter( fileName: "src/config.txt"));

            //Music
            bw.write(String.valueOf(gp.music.volumeScale));
            bw.newLine();
            //SE
            bw.write(String.valueOf(gp.se.volumeScale));
            bw.newLine();

            bw.close();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
    1 usage  Thanh Nguyễn
    public void loadConfig(){

        try {
            BufferedReader br = new BufferedReader(new FileReader( fileName: "src/config.txt"));

            //Music
            String s = br.readLine();
            gp.music.volumeScale = Integer.parseInt(s);

            //SE
            s = br.readLine();
            gp.se.volumeScale = Integer.parseInt(s);

            br.close();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
}
```

- GamePanel: The GamePanel class represents the main game panel where the game logic and rendering take place. It's a crucial part of the game engine. This class has a lot of functionalities including game initialization, updating game state, rendering the game screen, managing game entities, handling input, and controlling the game loop. Its fields hold essential information about the game environment, player, entities, and various game states.

```
public GamePanel(){
    this.setPreferredSize(new Dimension(screenWidth,screenHeight));
    this.setBackground(Color.black);
    this.setDoubleBuffered(true);
    this.addKeyListener(keyH);
    this.setFocusable(true);
}

1 usage  Thanh Nguyễn +2
public void setupGame(){
    playMusic( i: 0);
    aSetter.setObject();
    aSetter.setSlime();
    aSetter.setBat();
    gameState = titleState;

    tempScreen = new BufferedImage(screenWidth,screenHeight,BufferedImage.TYPE_INT_ARGB);
    g2 = (Graphics2D) tempScreen.getGraphics();
    setFullScreen();
}
1 usage  Thanh Nguyễn
public void retry(){
    player.setDefaultPosition();
    player.restoreLife();
    aSetter.setBat();
    aSetter.setSlime();
}
3 usages  Thanh Nguyễn
public void restart(){
    player.setDefaultValues();
    player.setDefaultPosition();
    player.restoreLife();
    aSetter.setBat();
    aSetter.setSlime();
}
```

```java
public void setFullScreen(){
    //Set local screen device
    GraphicsEnvironment ge = GraphicsEnvironment.getLocalGraphicsEnvironment();
    GraphicsDevice gd = ge.getDefaultScreenDevice();
    gd.setFullScreenWindow(Main.window);

    //Get FullScreen width and height
    screenWidth2 = Main.window.getWidth();
    screenHeight2 = Main.window.getHeight();
}
1 usage  Thanh Nguyễn
public void startGameThread(){
    gameThread = new Thread( task: this);
    gameThread.start();
}
Thanh Nguyễn
@Override
public void run(){
    //"Delta" method for GameLoop
    //60 FPS
    double drawInterval = (double) 1000000000 /FPS; //0.01666s
    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();
            drawToTempScreen(); //G2 draw image to the tempScreen
            drawToScreen(); // draw image from the tempScreen to the screen
            delta--;
        }
    }
}
```

```java
public void update(){
    if(gameState == playState){
        //player
        player.update();
        //bat
        for(int i = 0; i < bat.length; i++){
            if(bat[i] != null){
                if(bat[i].alive == true && bat[i].dying== false) {
                    bat[i].update();
                }
                if(bat[i].alive == false) {
                    bat[i] = null;
                }
            }
        }
        //slime
        for(int i = 0; i < slime.length; i++){
            if(slime[i] != null){
                if(slime[i].alive == true && slime[i].dying== false) {
                    slime[i].update();
                }
                if(slime[i].alive == false) {
                    slime[i] = null;
                }
            }
        }
        checkWinCondition();
    }
}
1 usage  Thanh Nguyễn +1
public void checkWinCondition(){
    boolean allBatsDead = true;
    boolean allSlimesDead = true;
    // Check all bats
    for (Entity b : bat) {
        if (b != null && !b.dying) {
            allBatsDead = false;
            break;
        }
    }
```

```
    // Check all slimes
    for (Entity s : slime) {
        if (s != null && !s.dying) {
            allSlimesDead = false;
            break;
        }
    }
    // If all bats and slimes are dead, change the game state to win state
    if (allBatsDead && allSlimesDead) {
        gameState = winState;
        stopMusic();
        playSE( i: 10);
    }
}
1 usage   nghianguyen19 +2
public void drawToTempScreen() {
    // title screen
    if (gameState == titleState) {
        ui.draw(g2);
    }
    // others
    else {
        // tile
        tileM.draw(g2);
        // object
        entityList.add(player);
        for (int i = 0; i < obj.length; i++) {
            if (obj[i] != null) {
                obj[i].draw(g2,  gp: this);
            }
        }
        //bat
        for (int i = 0; i < bat.length; i++) {
            if (bat[i] != null) {
                bat[i].draw(g2);
            }
        }
```

```
        //monster
        for (int i = 0; i < slime.length; i++) {
            if (slime[i] != null) {
                slime[i].draw(g2);
            }
        }
        // player
        player.draw(g2);
        //Draw Entity List
        for (int i = 0; i < entityList.size(); i++) {
            entityList.get(i).draw(g2);
        }
        //Empty entity list
        entityList.clear();
        // UI
        ui.draw(g2);
    }
}
1 usage  Thanh Nguyễn
public void drawToScreen() {
    Graphics g = getGraphics();
    g.drawImage(tempScreen, x: 0, y: 0,screenWidth2,screenHeight2, observer: null);
    g.dispose();
}
6 usages  Thanh Nguyễn
public void playMusic ( int i){
    music.setFile(i);
    music.play();
    music.loop();
}
5 usages  Thanh Nguyễn
public void stopMusic() { music.stop(); }
26 usages  Thanh Nguyễn
public void playSE ( int i){
    se.setFile(i);
    se.play();
}
```

- KeyHandler: is used to check whenever a key is pressed or released.

```
public KeyHandler(GamePanel gp) { this.gp = gp; }
 Thanh Nguyễn
@Override
public void keyTyped(KeyEvent e) {    }

 Thanh Nguyễn +1
@Override
public void keyPressed(KeyEvent e) {
    int code = e.getKeyCode();
    //title state
    if(gp.gameState == gp.titleState){titleState(code);}
    //play state
    else if(gp.gameState == gp.playState){playState(code);}
    //Pause State
    else if(gp.gameState == gp.pauseState){pauseState(code);}
    //Option State
    else if(gp.gameState == gp.optionsState) {optionState(code);}
    //Game Over State
    else if(gp.gameState == gp.gameOverState) {gameOverState(code);}
    //Win State
    else if(gp.gameState == gp.winState) {winState(code);}
}
```

```
public void titleState(int code){
    if ((code == KeyEvent.VK_W)) {
        gp.ui.commandNum--;
        gp.playSE( i: 6);
        if(gp.ui.commandNum < 0){
            gp.ui.commandNum = 1;
        }
    }
    if ((code == KeyEvent.VK_S)) {
        gp.ui.commandNum++;
        gp.playSE( i: 6);
        if(gp.ui.commandNum > 1){
            gp.ui.commandNum = 0;
        }
    }
    if(code == KeyEvent.VK_ENTER){
        if(gp.ui.commandNum == 0){
            gp.playSE( i: 7);
            gp.gameState = gp.playState;
            gp.stopMusic();
            gp.playMusic( i: 8);
        }
        if (gp.ui.commandNum == 1) {
            gp.playSE( i: 7);
            System.exit( status: 0);
        }
    }
}
```

\

```java
public void playState(int code){
    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_P){
        gp.playSE( i: 6);
        gp.gameState = gp.pauseState;
    }
    if(code == KeyEvent.VK_ENTER){
        enterPressed = true;
    }
    if(code == KeyEvent.VK_ESCAPE){
        gp.playSE( i: 6);
        gp.gameState = gp.optionsState;
    }
}
1 usage  Thanh Nguyễn +1
public void pauseState(int code){
    if(code == KeyEvent.VK_P){
        gp.gameState = gp.playState;
    }
}
```

```
public void optionState(int code){
    if(code == KeyEvent.VK_ESCAPE){
        gp.gameState = gp.optionsState;
    }
    if(code == KeyEvent.VK_ENTER){
        enterPressed = true;
    }
    int maxCommandNum = 0;
    switch (gp.ui.subState){
        case 0: maxCommandNum = 3;
    }
    if (code == KeyEvent.VK_W) {
        gp.ui.commandNum--;
        gp.playSE( i: 6);
        if (gp.ui.commandNum < 0) {
            gp.ui.commandNum = maxCommandNum;
        }
    }
    if (code == KeyEvent.VK_S) {
        gp.ui.commandNum++;
        gp.playSE( i: 6);
        if (gp.ui.commandNum > maxCommandNum) {
            gp.ui.commandNum = 0;
        }
    }
    if(code == KeyEvent.VK_A){
        if(gp.ui.commandNum == 0 && gp.music.volumeScale > 0){
            gp.music.volumeScale--;
            gp.music.checkVolume();
            gp.playSE( i: 7);
        }
        if(gp.ui.commandNum == 1 && gp.se.volumeScale > 0){
            gp.se.volumeScale--;
            gp.playSE( i: 7);
        }
    }
    if(code == KeyEvent.VK_D){
        if(gp.ui.commandNum == 0 && gp.music.volumeScale < 5){
            gp.music.volumeScale++;
            gp.music.checkVolume();
            gp.playSE( i: 7);
        }
```

```
        if(gp.ui.commandNum == 1 && gp.se.volumeScale < 5){
            gp.se.volumeScale++;
            gp.playSE( i: 7);
        }
    }
}
1 usage   Thanh Nguyễn
public void gameOverState(int code){
    if (code == KeyEvent.VK_W) {
        gp.ui.commandNum--;
        if (gp.ui.commandNum < 0) {
            gp.ui.commandNum = 1;
        }
        gp.playSE( i: 6);
    }
    if (code == KeyEvent.VK_S) {
        gp.ui.commandNum++;
        if(gp.ui.commandNum > 1) {
            gp.ui.commandNum = 0;
        }
        gp.playSE( i: 6);
    }
    if(code == KeyEvent.VK_ENTER){
        if(gp.ui.commandNum == 0){
            gp.gameState = gp.playState;
            gp.playSE( i: 7);
            gp.retry();
            gp.playMusic( i: 8);
        } else if (gp.ui.commandNum == 1) {
            gp.gameState = gp.titleState;
            gp.playSE( i: 7);
            gp.playMusic( i: 0);
            gp.restart();
        }
    }
}
```

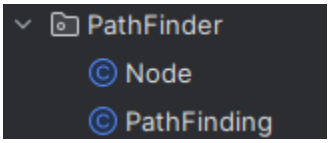
```java
public void winState(int code){
    if(code == KeyEvent.VK_ENTER){
        if (gp.ui.commandNum == 0) {
            gp.gameState = gp.titleState;
            gp.stopMusic();
            gp.playSE( i: 7);
            gp.playMusic( i: 0);
            gp.restart();
        }
    }
}
```

Thanh Nguyễn

```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;
    }
}
```

- Main: The Main class serves as the entry point for the game. It creates the game window, initializes the game panel, and starts the game loop.

```java
public class Main {
    12 usages
    public static JFrame window;
    Thanh Nguyễn +1
    public static void main(String[] args){
        window = new JFrame();
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        window.setResizable(false);
        window.setTitle("Last Time Adventure");

        GamePanel gamePanel = new GamePanel();
        window.add(gamePanel);

        gamePanel.config.loadConfig();
        window.setUndecorated(true);

        window.pack();

        window.setLocationRelativeTo(null);
        window.setVisible(true);

        gamePanel.setupGame();
        gamePanel.startGameThread();

    }
}
```



- Node: The Node class represents a node in the pathfinding grid.

```
public class Node {
    2 usages
    Node parent;
    7 usages
    public int col;
    7 usages
    public int row;
    4 usages
    int gCost;
    2 usages
    int hCost;
    4 usages
    int fCost;
    3 usages
    boolean solid;
    3 usages
    boolean open;
    3 usages
    boolean checked;
    1 usage  nghianguyen19
    public Node(int col,int row){
        this.col = col;
        this.row = row;
    }
}
```

- PathFinding: The PathFinding class encapsulates the logic for finding the shortest path from a start node to a goal node using the A* algorithm.

```
public PathFinding(GamePanel gp){
    this.gp = gp;
    instantiateNodes();
}
1 usage   nghianguyen19
public void instantiateNodes(){
    node = new Node[gp.maxWorldCol][gp.maxWorldRow];
    int col = 0;
    int row = 0;
    while(col < gp.maxWorldCol && row < gp.maxWorldRow){
        node[col][row] = new Node(col,row);
        col++;
        if(col == gp.maxWorldCol){
            col = 0;
            row++;
        }
    }
}
1 usage   nghianguyen19
public void resetNodes(){

    int col = 0;
    int row = 0;
    while(col < gp.maxWorldCol && row < gp.maxWorldRow){
        //Reset open, checked and solid state
        node[col][row].open = false;
        node[col][row].checked = false;
        node[col][row].solid = false;
        col++;
        if(col == gp.maxWorldCol){
            col = 0;
            row++;
        }
    }
    //Reset other settings
    openList.clear();
    pathList.clear();
    goalReached = false;
    step = 0;
}
```

```java
public void setNodes(int startCol, int startRow, int goalCol, int goalRow){
    resetNodes();
    //Set start and goal node
    startNode = node[startCol][startRow];
    currentNode = startNode;
    goalNode = node[goalCol][goalRow];
    openList.add(currentNode);

    int col = 0;
    int row = 0;
    while(col < gp.maxWorldCol && row < gp.maxWorldRow){
    //Set solid node
        //Check tiles
        int tileNum = gp.tileM.mapTileNum[col][row];
        if (gp.tileM.tile[tileNum].collision == true){
            node[col][row].solid = true;
        }
        //Set cost
        getCost(node[col][row]);
        col++;
        if(col == gp.maxWorldCol){
            col = 0;
            row++;
        }
    }
}
```

```java
public void getCost(Node node){
    //G cost
    int xDistance = Math.abs(node.col - startNode.col);
    int yDistance = Math.abs(node.row - startNode.row);
    node.gCost = xDistance + yDistance;
    //H cost
    xDistance = Math.abs(node.col - goalNode.col);
    yDistance = Math.abs(node.row - goalNode.row);
    node.hCost = xDistance + yDistance;
    //F cost
    node.fCost = node.gCost + node.hCost;
}
1 usage  nghianguyen19
public boolean search(){
    while (goalReached == false && step < 500){
        int col = currentNode.col;
        int row = currentNode.row;

        //Check the currentNode
        currentNode.checked = true;
        openList.remove(currentNode);

        //Open the up node
        if(row - 1 >= 0){
            openNode(node[col][row-1]);
        }
        //Open the left node
        if(col - 1 >= 0){
            openNode(node[col-1][row]);
        }
        //Open the down node
        if(row + 1 < gp.maxWorldRow){
            openNode(node[col][row+1]);
        }
        //Open the right node
        if(col + 1 < gp.maxWorldCol){
            openNode(node[col+1][row]);
        }
```

```
        //Find the best Node
        int bestNodeIndex = 0;
        int bestNodefCost = 999;
        for(int i = 0; i < openList.size(); i++){
            //Check if this node's F cost is better
            if(openList.get(i).fCost < bestNodefCost){
                bestNodeIndex = i;
                bestNodefCost = openList.get(i).fCost;
            }
            //If F cost is equal, check the G cost
            else if(openList.get(i).fCost == bestNodefCost){
                if (openList.get(i).gCost < openList.get(bestNodeIndex).gCost){
                    bestNodeIndex = i;
                }
            }
        }

        //If there is no node in the openList, end the loop
        if(openList.size() == 0){
            break;
        }

        //After the loop, openList[bestNodeIndex] is the next step (= currentNode)
        currentNode = openList.get(bestNodeIndex);
        if (currentNode == goalNode){
            goalReached = true;
            trackThePath();
        }
        step++;
    }
    return goalReached;
}
```

```
public void openNode(Node node){
    if(node.open == false && node.checked == false && node.solid == false){
        node.open = true;
        node.parent = currentNode;
        openList.add(node);
    }
}
1 usage  nghianguyen19
public void trackThePath(){
    Node current = goalNode;
    while(current != startNode){
        pathList.add( index: 0,current);
        current = current.parent;
    }
}
```

CHAPTER 4: FINAL GAME

Source code:

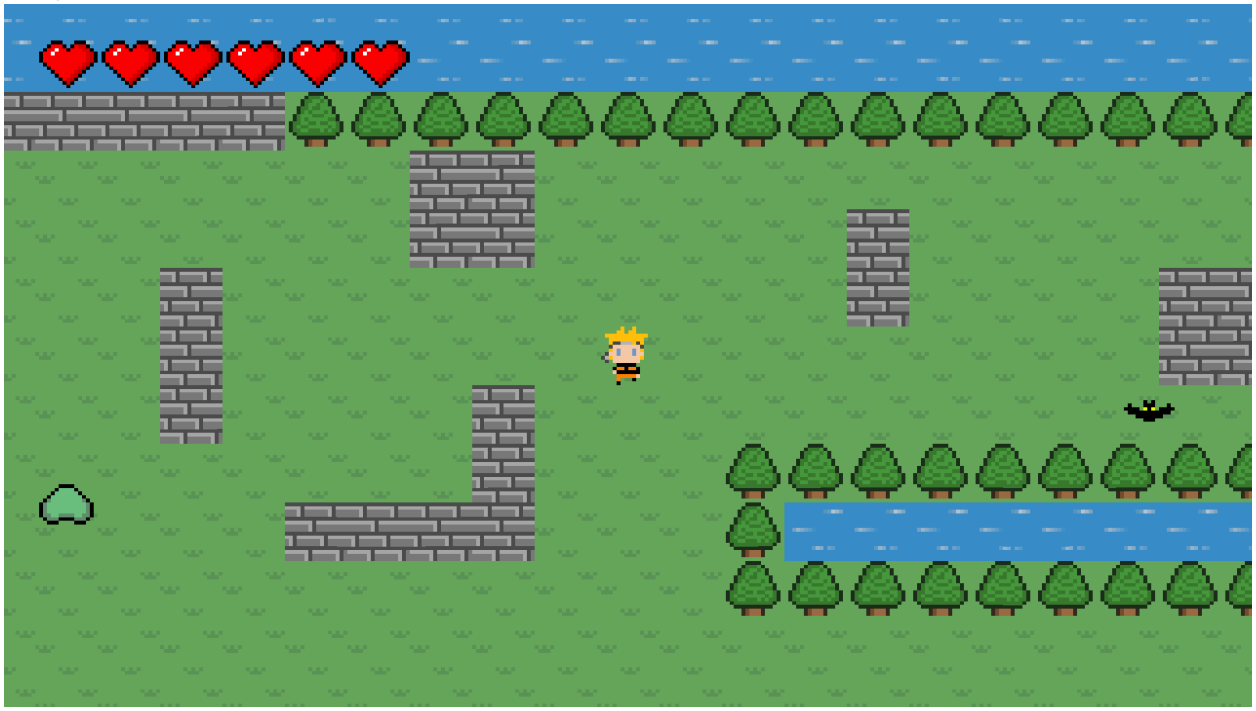
https://github.com/nguyeenphan/LastTimeAdventure

Instruction gameplay:

Title screen:

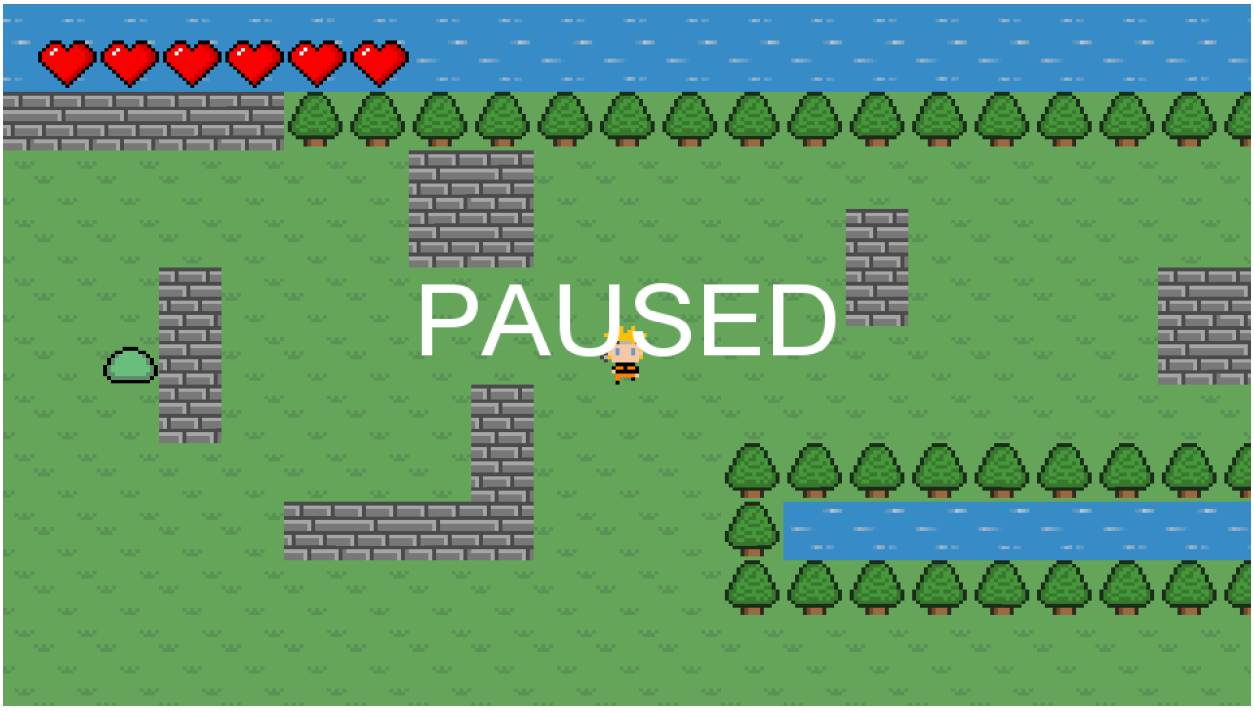
In this screen, you can choose to play a new game by pressing ENTER when the arrow is next to the NEW GAME option. If you want to exit the game, you can press ENTER on the QUIT option. To switch between NEW GAME and QUIT, press W or S to cycle between the options.

Play screen:

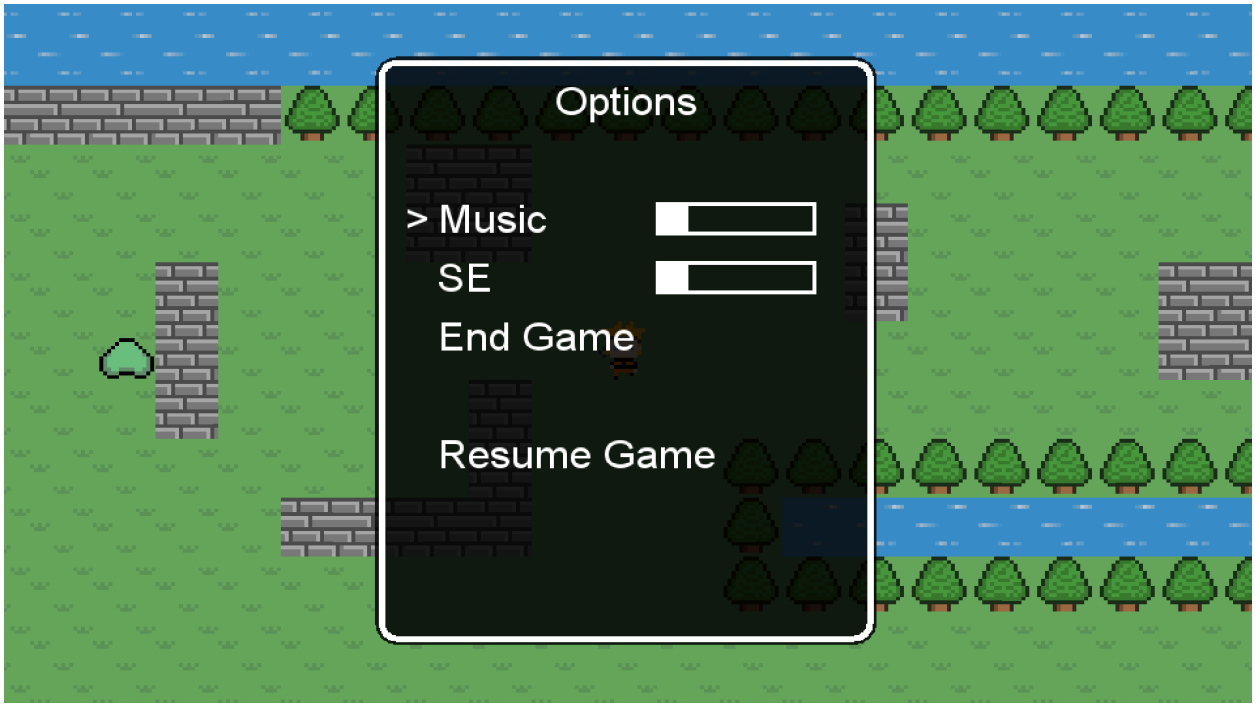


This is where you can play the game. To make the character move, press W, A, S, D to move to its corresponding direction and press ENTER to make the character attack. To win the game, you need to go around the map and defeat every monster.

Pause screen :



Options screen:



Press ESC to open the Options Screen. This is where you can change the volume of the music or the sound effect, and if you want to return to the title screen, press ENTER on End Game. To return to the game, press ENTER on Resume Game.

Game over screen:

This screen will appear when you run out of health before defeating all the monsters. If you want to try again, press ENTER on Retry; or if you want to quit the game, press ENTER on Quit.

Win screen:

If you reach this screen, congratulations! You win! If you want to play the game again or exit, press ENTER on Back To Title Screen.

CHAPTER 5: EXPERIENCE

"Last Time Adventure" represents the culmination of our efforts to create a compelling and engaging 2D top-down wave-based survival game. This project has allowed us to apply and expand our skills in game development, including programming, design, and project management. The game combines the nostalgic charm of 8-bit graphics with modern gameplay mechanics to provide a unique and enjoyable experience for players.

"Last Time Adventure" stands as a testament to our dedication and collaboration. It is a game that pays homage to the classic era of gaming while introducing new elements that keep the gameplay fresh and exciting. We are proud of what we have accomplished and believe that this project will serve as a strong foundation for our future endeavors in game development. We hope players will find as much joy in playing "Last Time Adventure" as we did in creating it.