INTERNATIONAL UNIVERSITY

VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY

School of Computer Science and Engineering

-----*-----**

PROJECT REPORT

SAVING SIR. NGHIA

Advisor: Dr Tran Thanh Tung and Nguyen Trung Nghia

Course: Object – Oriented Programming

No.	Full Name	Student's ID
1	Đỗ Thanh Bảo Anh	ITDSIU22175
2	Phạm Tuấn Đăng Khoa	ITITIU22087
3	Nguyễn Minh Kha	ITITIU22079
4	Trần Thiên Phú	ITITIU22124

TABLE OF CONTENTS

ABSTRACT

This report focuses on the development of a 2D Top-Down RPG desktop game, "Saving Sir. Nghia." Inspired by Soul Knight [1], the project features various dungeons for players to explore. It incorporates a player experience system with three distinct character skins and intelligent, computer-controlled enemies – both melee and skill-based – to challenge players. Unlike traditional Top-Down RPGs where players attack with swords, this game emphasizes searching for keys and unlocking doors. "Saving Sir. Nghia" explores a new dimension in traditional RPGs by seamlessly blending puzzle-solving with survival elements, where players must defeat enemies to survive. With its simplicity, the game aims to recapture the fun of classic titles like Roguelike, Pokémon, and Mario, while incorporating modern features.

Keywords: dungeon, top down, adventurous, game 2D, object-oriented programming.

CHAPTER 1: INTRODUCTION

1. Objectives

The goal of the project is to create a 2D game based on top-down game concepts, knowledge of the Java programming language, and basic available design patterns. The game also demonstrated the strengths of object-oriented programming methods in creating games. This is a quite classic game with simple steps, does not require high skills or complexity in the gameplay but still brings an attractive feeling, making players easily attracted to the rhythm. game level. Saving Sir. Nghia allows us to experience the feeling of the player himself as a companion of the main character, accompanying the character through dangerous areas, fighting with monsters, and It's time to find a way to unlock dangerous organs to free yourself. With the efforts of our team members, we have created a program that can provide users with extremely enjoyable experiences. Additionally, we will also try to fix and add more features in the future.

To put it more succinctly, the purpose of the project is

- Helps us better understand Game Development Concepts such as game loops, collision detection, rendering, and game physics...
- Apply knowledge of Java programming language and Object-Oriented Programming to create a game that can bring attraction and entertainment to players.
- Experience the processes of ideating, creating, managing, and improving games.

2. The Tools Used

- IDE for programming and debugging: IntelliJ, VSCode, Eclipse.
- Design: Piskel, Simple2DTileEditor.
- Java Development Kit: 21.
- Mean of code version management: GitHub.
- Means of contacting: Facebook

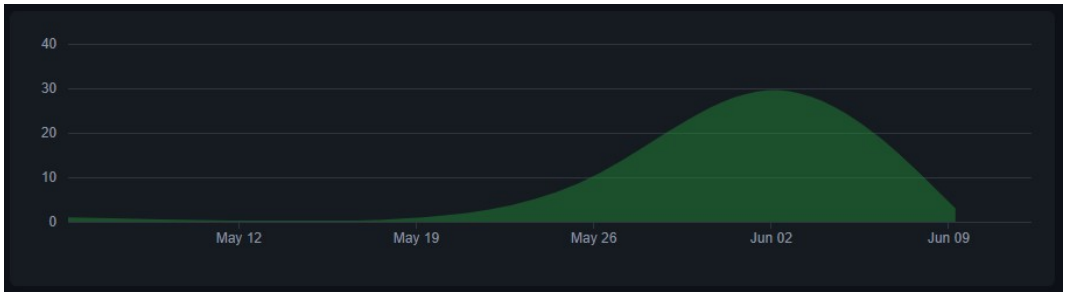

Figure 1: GitHub statistics

3. Developer Team

Name	Contribute
Đỗ Thanh Bảo Anh	➢ Write Report, Slide, Presentation ➢ Develop Code: Class Boss, Screen Level Up,... ➢ Fix Bug
Phạm Tuấn Đăng Khoa	➢ Write Report, Slide, Presentation ➢ Design Map ➢ Develop Code ➢ Fix Bug
Nguyễn Minh Kha	➢ Write Report, Slide, Presentation ➢ Find Asset ➢ Design Player Character, NPC,... ➢ Develop Code: UI, Key System,..
Trần Thiên Phú	➢ Write Report, Slide, Presentation ➢ Develop Code: UI, Torch,... ➢ Fix Bug

CHAPTER 2: SYSTERM DESIGN

1. Package Structure

The project's current structure is the result of extensive iteration and troubleshooting to arrive at this optimized configuration, as the carefully crafted project layout enables the seamless execution of the game algorithms powering our latest creation and driving the interactive experiences in our game.

To facilitate the management of the classes, we have organized them into distinct groups, like:

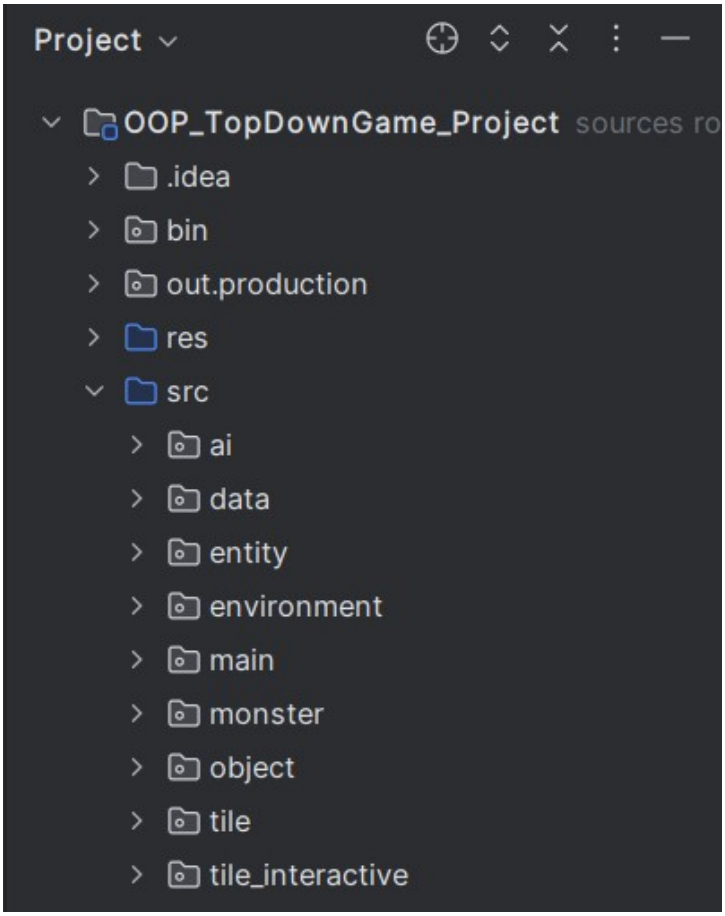

Figure 2: Package Structure

Here is the link to the GitHub repository for our project: https://github.com/Khoataphat/OOP_TopDownGame_Project

2. List of Class and Responsibility

Package	Class Name	Responsibility
ai	Node	provides functionality to represent and manage individual nodes in a grid-based pathfinding

		algorithm.
	PathFinder	is responsible for finding a path between two points in the game world. It uses a grid-based pathfinding algorithm to navigate through the game's tiles and obstacles.
data	Progress	store the progress of the game in a static variable called skeletonLordDefeated.
entity	Entity	serves as a representation of various entities within a game world, such as characters, NPCs (non-playable characters), or objects. It encapsulates the necessary features and properties to accurately portray these entities in the game.
	NPC_Merchant	which extends the "Entity" class, represents a merchant NPC (non-playable character) in the game. This class is responsible for defining the behavior and attributes of a merchant NPC, including their dialogue options, inventory of items, and interaction functionality, enriching the game world with trade opportunities.
	NPC_OldMan	which extends the "Entity" class, represents an instructor NPC (non-playable character) in the game. This class is specifically designed to define the behavior and attributes of an instructor NPC, providing necessary functionalities to provide helpful guidance to players.
	Particle	which extends the "Entity" class, provides the necessary functionalities to create and manage particles in the game, allowing for dynamic visual effects such as sparks.
	Player	which extends the "Entity" class, is the embodiment of the hero in our grand adventure. It encapsulates the behavior and attributes of the player, providing various functionalities to handle movement, interaction, and other actions.
	PlayerDummy	which extends the "Entity" class, represents a dummy player entity in the game. It serves as a placeholder or test entity during development and

		does not represent the actual player-controlled character.
	Projectile	extends the "Entity" class, representing a projectile entity in the game which can be used for ranged attacks or as objects that can move through the game world.
environment	Lighting	is responsible for managing the lighting effects.
main	AssetSetter	implementing an asset setter for a game. It allows the user to set specific positions for initializing monsters within the game world.
	CollisionChecker	responsible for detecting and handling collisions between entities, objects, and the player.
	Config	is designed to manage the game's configuration settings, such as fullscreen mode and volume levels for music and sound effects.
	CutsceneManager	is responsible for managing cutscenes in our game. It handles the sequence of events, animations, and transitions that occur during a cutscene.
	EntityGenerator	is responsible for generating different types of Entity objects based on the given itemName.
	EvenHandler	handles different events and interactions in the game, such as set Dialogue, speak, and hit.
	EvenRect	initialize a rectangular region in the game to define the boundary of the event region to support collision detection with the player's solid region.
	GamePanel	inherits from the JPanel class and implements the Runnable interface, is responsible for managing and displaying the game interface. It handles game states, updates components, and renders game elements on the screen.
	KeyHandler	handles key events in our game. It implements the KeyListener interface, which allows it to listen for and respond to key presses, releases, and typed

		events.
	Main	sets up the main game window, configures its properties, and starts the game loop to begin gameplay.
	Sound	handle sound effects in our game. It provides methods for loading and playing various sound files.
	UI	responsible for handling various graphical user interface elements and interactions in the game.
	Utility Tool	responsible for scaling images in our game, allowing for easy resizing and manipulation of images for various purposes.
monster	MON_Bat	extends the "Entity" class,responsible for defining the behavior and attributes of a specific type of monster called "Bat" in the game.
	MON_GreenSlime	extends the "Entity" class,responsible for defining the behavior and attributes of a specific type of monster called "GreenSlime" in the game.
	MON_Orc	extends the "Entity" class,responsible for defining the behavior and attributes of a specific type of monster called "Orc" in the game.
	MON_RedSLime	extends the "Entity" class,responsible for defining the behavior and attributes of a specific type of monster called "RedSlime" in the game.
	MON_SkeletonLord	Extends the “Entity” class, responsible for defining the behavior and attributes of the final boss.
object	OBJ_Chest	represents a treasure chest in your game, handling contents with player interaction, and its open/closed state.
	OBJ_Coin_Bronze	represents a bronze coin in your game, handling its value, price, and the action of using it to increase the player's coin count.
	OBJ_Door	represents a locked door in your game, requiring a

		key to open and displaying a message to the player.
	OBJ_Door_Iron	represents an unbreakable iron door in your game, displaying a message that it cannot be opened.
	OBJ_Fireball	represents a fireball projectile of player.
	OBJ_Heart	represents a heart item in your game, handling its value and the action of using it to increase the player's health.
	OBJ_Key	represents a key item in your game, handling its use to open doors and displaying messages accordingly.
	OBJ_Lantern	class represents a lantern in game.
	OBJ_ManaCrystal	represents a mana crystal item in your game, handling its value and the action of using it to increase the player's mana.
	OBJ_Potion_Red	represents a red potion in your game, handling its use to heal the player and displaying a message confirming the health increase.
	OBJ_Rock	represents a rock projectile of monster.
	OBJ_Sword_Normal	represents a normal sword in your game, defining its properties like attack power, attack area, and animation durations.
	OBJ_thay_nghia	represents an item related to the character "Teacher Nghia". When interacted with, it triggers the ending scene.
tile	Map	provides functionality for creating, managing, and displaying the game map in both full screen and mini map modes.
	Tile	represents individual tiles with their visual appearance and collision status

	TileManage	manages the tiles used in the game, including loading tile data, setting up tiles, loading map data, and drawing the tiles on the game panel.
tile_interactive	Interactive	is a subclass of the "Entity" class and represents interactive tiles in the game world.
	IT_AreaAttack	is a subclass of the "InteractiveTile" class and represents an interactive tile called "Area Attack" in the game world.

Table 1: List of class and responsibility

3. UML Diagram

We provided the UML diagrams for the entire project and each group that was addressed to help you better understand the structure and methods. Whole UML Diagram: https://github.com/Khoataphat/OOP_TopDownGame_Project/blob/91b688f75caeb4fb7a783d08951a4c38c9cef223/UML_OOP_Project.pdf

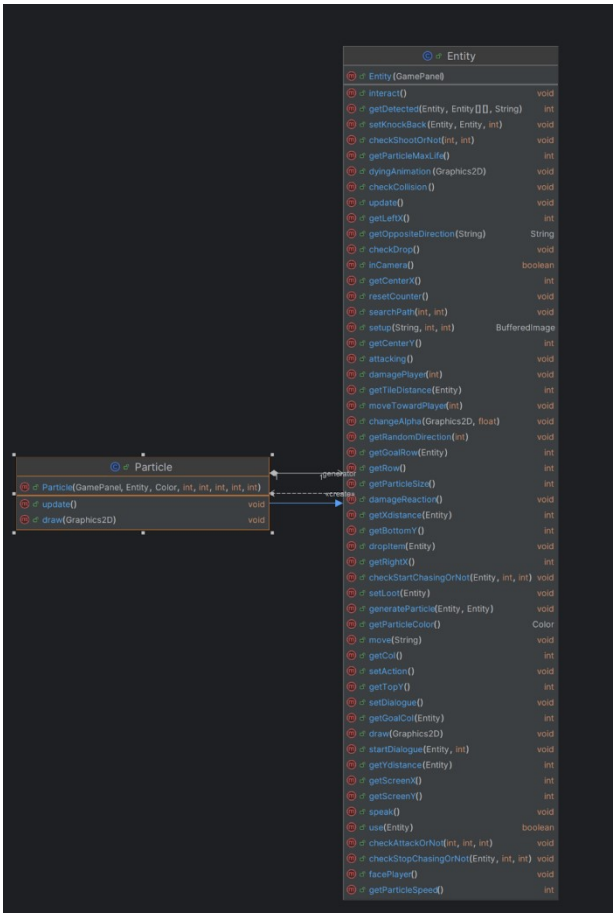

Figure 3: Entity diagram

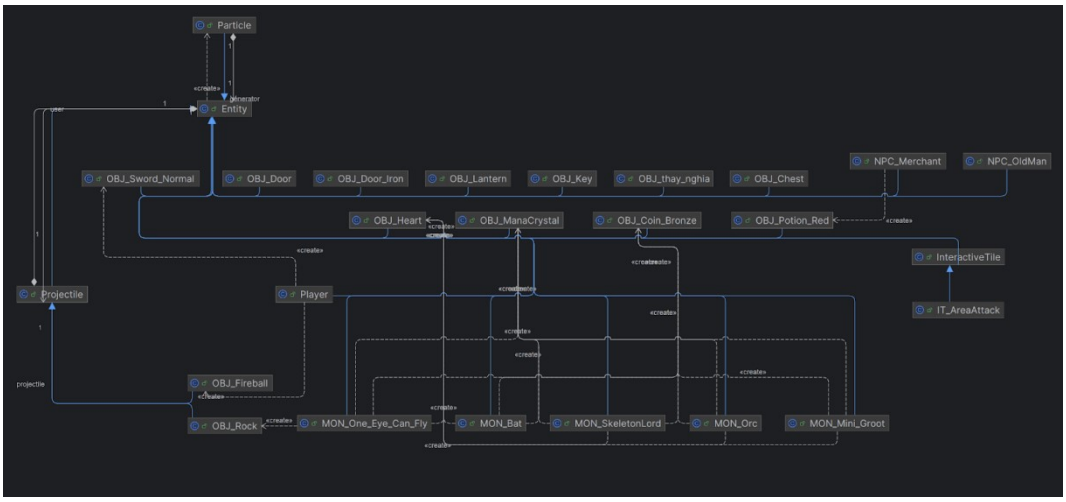

Figure 4: Class extend from Entity diagram

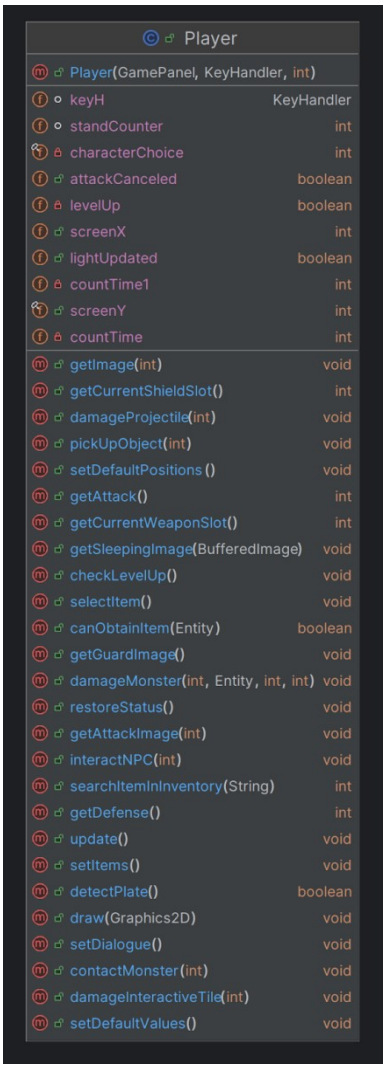

Figure 5: Player diagram

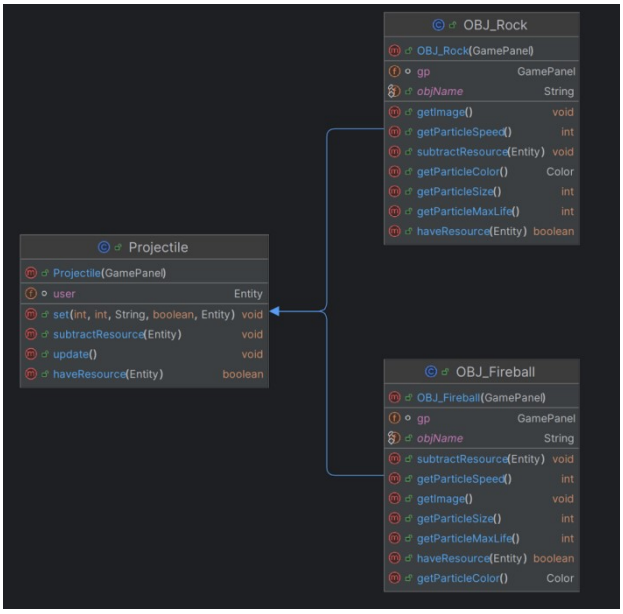

Figure 6: Projectile diagram

Figure 7: Object diagram

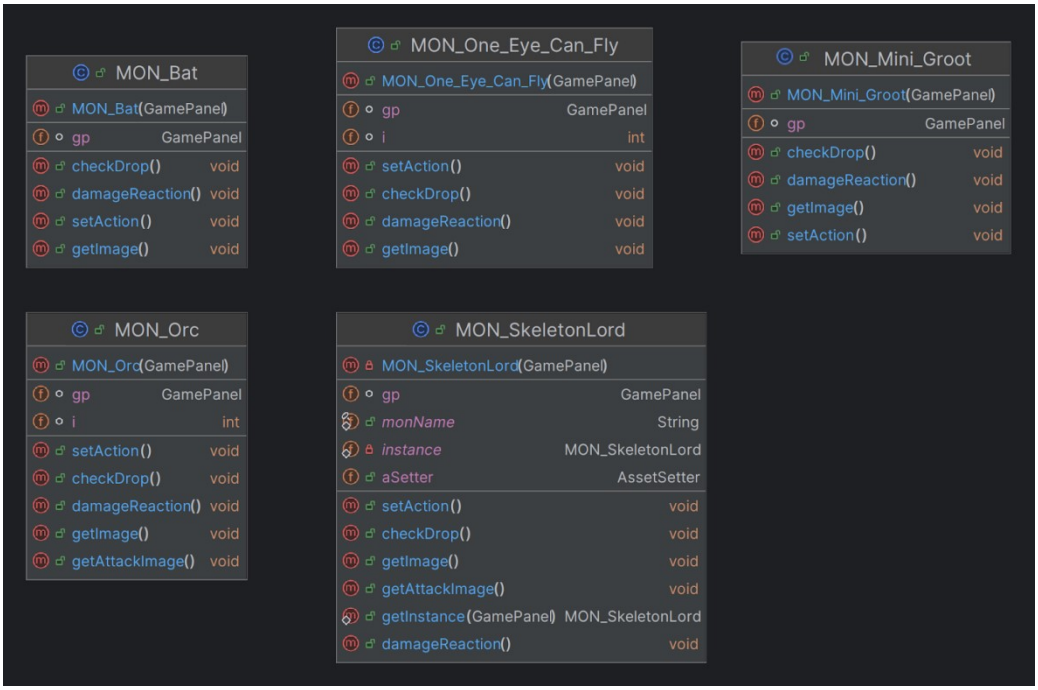

Figure 8: Monster diagram

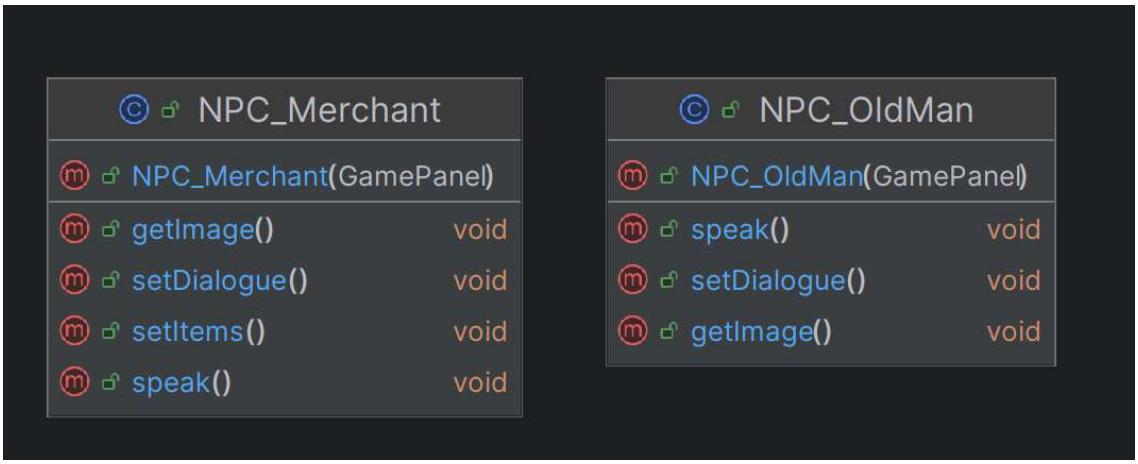

Figure 9: NPC diagram

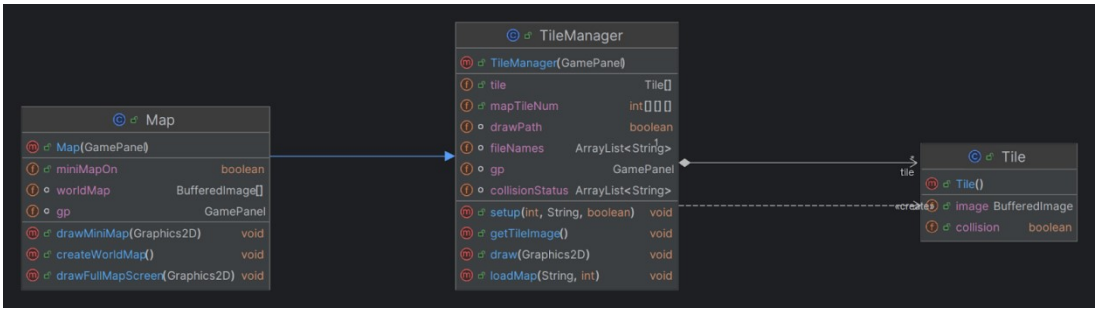

Figure 10: Map diagram

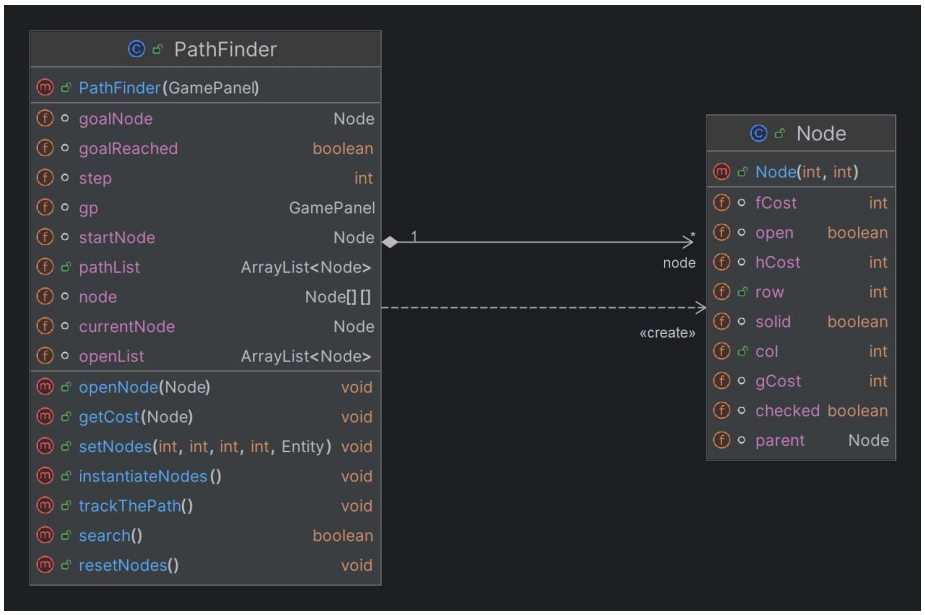

Figure 11: Path finding algorithm diagram

Figure 12. InteractiveTile diagram

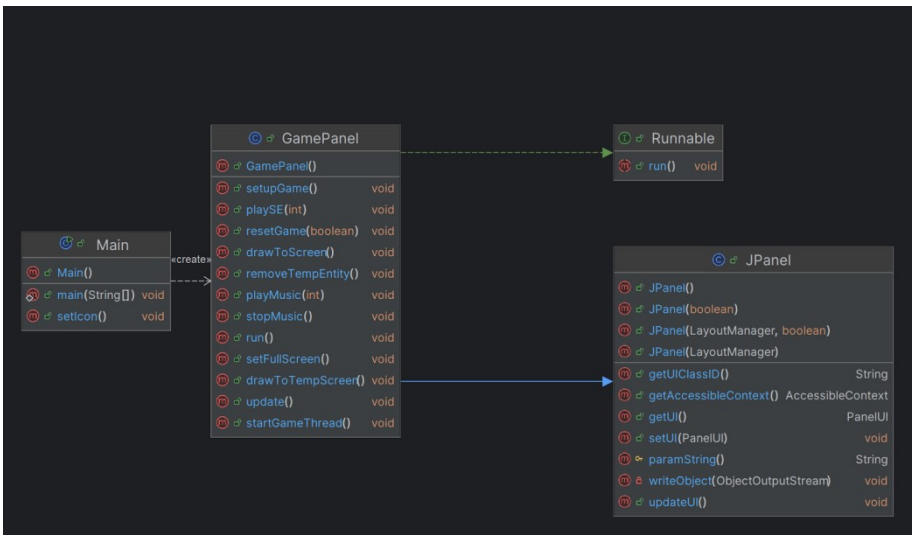

Figure 13: Main diagram

Figure 14: Panel diagram

4. Code Flow

Start from class Main, create a screen, and initialize GamePanel. In Main, call the setupGame method of GamePanel to set up the game before it starts. Then call the startGameThread to start the game in GamePanel.

```java
public class Main {  Khoa Tap Hat +1 *

    public static JFrame window;  13 usages

    public static void main(String[] args) {  Khoa Tap Hat +1
        window = new JFrame();
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        window.setResizable(false); // Cant Resizable
        window.setTitle("Saving Sir.Nghia\n"); // Window Name
        new Main().setIcon();
        GamePanel gamePanel = new GamePanel();
        window.add(gamePanel);

        //gamePanel.config.loadConfig();
        if(gamePanel.fullScreenOn == true)
        {
            window.setUndecorated(true);
        }

        window.pack(); // Resizes to prefered size and prevents overflow.

        window.setLocationRelativeTo(null); // Starts center of screen
        window.setVisible(true);

        gamePanel.setupGame(); // Setting up the game before starts
        gamePanel.startGameThread();
    }
```

Figure 15: Main function

```java
public void setupGame()  1 usage  Khoa Tap Hat +1
{
    aSetter.setObject();
    aSetter.setNPC();
    aSetter.setMonster();
    aSetter.setInteractiveTile();
    //The interface
    lighting.setup();

//      eManager.setup();

    /*playMusic(0);   // 0 = BlueBoyAdventure.wav
    stopMusic();*/
    gameState = titleState;
    //FOR FULLSCREEN
    tempScreen = new BufferedImage(screenWidth,screenHeight,BufferedImage.TYPE_INT_ARGB); //blank screen
    g2 = (Graphics2D) tempScreen.getGraphics(); // g2 attached to this tempScreen. g2 will draw on this tempScreen buffered image.
    if(fullScreenOn == true)
    {
        setFullScreen();
    }
}
```

Figure 16: setupGame function in GamePanel

In GamePanel, initialize values and lists to run the game in the run method to display on the screen.

```java
    @Override  Khoa Tap Hat *
    public void run()
    {
        double drawInterval = 1000000000/FPS;
        double delta = 0;
        long lastTime = System.nanoTime();
        long currentTime;

        while(gameThread != null)
        {
            currentTime = System.nanoTime();

            delta += (currentTime - lastTime) / drawInterval;
            lastTime = currentTime;
            if(delta >= 1)
            {
                update();
                drawToTempScreen(); //FOR FULL SCREEN - Draw everything to the buffered image
                drawToScreen();     //FOR FULL SCREEN - Draw the buffered image to the screen
                delta--;
            }
        }
    }
```

Figure 17: run function in GamePanel

In the run method, call update and drawToTempScreen to create a game loop. This loop will update information about the player class, monster, game state, and so on in the update method, and it will continuously draw the game on the screen with drawToTempScreen.

```
public void update()  1 usage   Khoa Tap Hat +1
{
    if(gameState == playState)
    {...}

    if(gameState == pauseState)
    {
        //nothing, just pause screen
    }
}
```

Figure 18: update function in GamePanel

```
public void drawToTempScreen()  1 usage   Khoa Tap Hat +1 *
{
    /*//DEBUG
    long drawStart = 0;
    if(keyH.showDebugText == true)
    {
        drawStart = System.nanoTime();
    }

     */

    //TITLE SCREEN
    if(gameState == titleState)
    {
        ui.draw(g2);
    }
    //MAP SCREEN
    else if(gameState == mapState)
    {
        map.drawFullMapScreen(g2);
    }
    //OTHERS
    else
    {...}
}
```

Figure 19: drawToTempScreen function in GamePanel

```
//GAME STATE
public int gameState;  58 usages
public final int titleState = 0;  6 usages
public final int playState = 1;  18 usages
public final int pauseState = 2;  4 usages
public final int dialogueState = 3;  6 usages
public final int characterState = 4;  3 usages
public final int optionsState = 5;  3 usages
public final int gameOverState = 6;  3 usages
public final int transitionState = 7;  no usages
public final int tradeState = 8;  3 usages
public final int sleepState = 9;  1 usage
public final int mapState = 10;  3 usages
public final int cutsceneState = 11;  5 usages
public final int levelupState = 12;  3 usages
```

Figure 20: Game State

When gameState is titleState, it calls drawTitleScreen to draw the first Title on the screen and keyHandler to handle the button, If you select play, it will switch to the second title screen to select the character.

```
if(titleScreenState == 0)
{

    //TITLE NAME
    g2.setFont(g2.getFont().deriveFont(Font.BOLD,  size: 96F));
    g2.drawImage(gp.player.mainInterface, x: 0, y: 0,gp.screenWidth,gp.screenHeight, observer: null);
    String text = "Saving Sir.Nghia\n";
    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);
    //MENU
    g2.setFont(g2.getFont().deriveFont(Font.BOLD,  size: 48F));

    text = "PLAY GAME";
    x = getXforCenteredText(text);
    y += gp.tileSize * 3.5;
    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);
    }
}
```

Figure 21: The 1st title screen in UI class

```
public void titleState(int code)  1 usage  Khoa Tap Hat *
{
    //MAIN MENU
    if (gp.ui.titleScreenState == 0) {
        if (code == KeyEvent.VK_W) {
            gp.ui.commandNum--;
            if (gp.ui.commandNum < 0) {
                gp.ui.commandNum = 2;
            }
        }
        if (code == KeyEvent.VK_S) {
            gp.ui.commandNum++;
            if (gp.ui.commandNum > 2) {
                gp.ui.commandNum = 0;
            }
        }
        if (code == KeyEvent.VK_ENTER) {
            if (gp.ui.commandNum == 0) {
                gp.ui.titleScreenState = 1; // Character class selection screen
                //gp.gameState = gp.playState;
            }

             */
            if (gp.ui.commandNum == 1) {
                System.exit( status: 0);
            }
        }
    }
```

Figure 22: The 1st title screen in KeyHandler class

Then the player selects a character, gameState changes to playState, and the KeyHandler retrieves the character's information and transmits it to the Player class. The Player class then uses this information to obtain the appropriate image.

```
else if (gp.ui.titleScreenState == 1) {
    if (code == KeyEvent.VK_W) {...}
    if (code == KeyEvent.VK_S) {...}

    if (code == KeyEvent.VK_ENTER) {
        //FIGHTER
        if (gp.ui.commandNum == 0) {
            System.out.println("Do some fighter specific stuff!");
            gp.player = new Player(gp, keyH: this, characterChoice: 1);
            gp.gameState = gp.playState;
            //Khoa
            gp.playMusic( i: 0);
        }
        //THIEF
        if (gp.ui.commandNum == 1) {
            System.out.println("Do some thief specific stuff!");
            gp.player = new Player(gp, keyH: this, characterChoice: 2);
            gp.gameState = gp.playState;
            //Khoa
            gp.playMusic( i: 0);
        }
        //SORCERER
        if (gp.ui.commandNum == 2) {
            System.out.println("Do some sorcerer specific stuff!");
            gp.player = new Player(gp, keyH: this, characterChoice: 3);
            gp.gameState = gp.playState;
            //Khoa
            gp.playMusic( i: 0);
        }
        //BACK
        if (gp.ui.commandNum == 3) {
            gp.ui.titleScreenState = 0;
        }
    }
}
```

Figure 23: The 2nd title screen in KeyHandler class

```
public Player(GamePanel gp, KeyHandler keyH, int characterChoice)  5 usages  Khoa Tap Hat +1
{
    super(gp); // calling constructor of super class(from entity class)
    this.keyH=keyH;
    this.characterChoice = characterChoice;
```

Figure 24: Constructor of Player class

```
public void getImage(int characterChoice )  Khoa Tap Hat +1
{
    switch(characterChoice){...}

}
```

Figure 25: getImage function of Player class

Subsequently, the Player class's update method runs, updating the player's state and animation.

When the game state changes to playState, the keyhandler will handle functions related to the game player such as moving the character, checkLevelup, ...

```
public void playState(int code)  1 usage  Khoa Tap Hat +1
{...}
public void pauseState(int code)  1 usage  Khoa Tap Hat
{...}
public void dialogueState(int code)  1 usage  Khoa Tap Hat
{...}
public void characterState(int code)  1 usage  Khoa Tap Hat
{...}
public void optionsState(int code)  1 usage  Khoa Tap Hat
{...}
public void gameOverState(int code)  1 usage  Khoa Tap Hat
{
```

Figure 26: Functions in playState and so on

When the player runs out of health, the gameState will switch to gameOverState and call the drawGameOverScreen function to draw the game over window, and the keyhandler to handle button information. If the player chooses to retry, the gameState will return to the playState state, continuing the game, If you choose to quit game, gameState will return the titleState and draw the original screen.

```
585         if(keyH.godModeOn == false)
586         {
587             if(life <= 0)
588             {
589                 gp.gameState = gp.gameOverState;
590                 gp.ui.commandNum =- 1; //for if you die while pressing enter
591                 gp.stopMusic();
592                 gp.playSE( i: 12);
593             }
```

Figure 27: Condition when the player runs out of health

```
public void gameOverState(int code)  1 usage  Khoa Tap Hat
{
    if(code == KeyEvent.VK_W)
    {
        gp.ui.commandNum--;
        if(gp.ui.commandNum < 0)
        {
            gp.ui.commandNum = 1;
        }
        gp.playSE( i: 9);
    }
    if(code == KeyEvent.VK_S)
    {
        gp.ui.commandNum++;
        if(gp.ui.commandNum > 1)
        {
            gp.ui.commandNum = 0;
        }
        gp.playSE( i: 9);
    }
    if(code == KeyEvent.VK_ENTER)
    {
        if(gp.ui.commandNum == 0) //RETRY, reset position, life, mana, monsters, npcs...
        {
            gp.gameState = gp.playState;
            gp.resetGame( restart: false);
            gp.playMusic( i: 0);
        }
        else if(gp.ui.commandNum == 1) //QUIT, reset everything
        {
            gp.ui.titleScreenState = 0;
            gp.gameState = gp.titleState;
            gp.resetGame( restart: true);
        }
    }
}
```

Figure 28: gameOverState function in KeyHandler class

```
public void drawGameOverScreen()  1 usage  Khoa Tap Hat
{
    g2.setColor(new Color( r: 0, g: 0, b: 0, a: 150)); //Half-black
    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);
    //Text
    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 THE TITLE SCREEN
    text = "Quit";
    x = getXforCenteredText(text);
    y += 55;
    g2.drawString(text,x,y);
    if(commandNum == 1)
    {
        g2.drawString( str: ">",  x: x-40, y);
    }
```

Figure 29: drawGameOverScreen function in UI class

When the player hits the checkEvent coordinates in the EventHandler class, the gameState will switch to cutsceneState. In the cutsceneState phase, start with drawing scene_skeletonLord in CutsceneManager, create objects that overlap the map to prevent the player from leaving and return the gameState state to playState.

```
public void checkEvent()  1 usage   Khoa Tap Hat
{
    //Check if the player character is more than 1 tile away from the last event
    int xDistance = Math.abs(gp.player.worldX - previousEventX);  //pure distance
    int yDistance = Math.abs(gp.player.worldY - previousEventY);
    int distance = Math.max(xDistance, yDistance);               //returns greater value
    if(distance > gp.tileSize)
    {
        canTouchEvent = true;
    }

    if(canTouchEvent == true)
    {
        //if(hit(0,23,12, "any") == true) {healingPool(gp.dialogueState);}
        //else if(hit(0,27,16, "right") == true) {damagePit(gp.dialogueState);}
        //else if(hit(0,10,39, "any") == true) {teleport(1,12,13,gp.indoor);} //to merchant's house
        //else if(hit(1,12,13, "any") == true) {teleport(0,10,39,gp.outside);} //to outside
        if(hit( map: 0, col: 42, row: 44, reqDirection: "any") == true) {speak(gp.npc[1][0]);} //merchant

        //else if(hit(0,12,9, "any") == true) {teleport(2,9,41,gp.dungeon);} //to the dungeon
        //else if(hit(2,9,41, "any") == true) {teleport(0,12,9,gp.outside);} //to outside
        //else if(hit(2,8,7, "any") == true) {teleport(3,26,41,gp.dungeon);} //to B2
        //else if(hit(3,26,41, "any") == true) {teleport(2,8,7,gp.dungeon);} //to B1
        else if(hit( map: 0, col: 47, row: 88, reqDirection: "any") == true) {skeletonLord();} //BOSS
        else if(hit( map: 0, col: 47, row: 89, reqDirection: "any") == true) {skeletonLord();} //BOSS
        else if(hit( map: 0, col: 47, row: 90, reqDirection: "any") == true) {skeletonLord();} //BOSS
    }

}
public boolean hit(int map, int col, int row, String reqDirection)  4 usages   Khoa Tap Hat
{...}
```

Figure 30: checkEvent function in EventHandler class

```
public void scene_skeletonLord()  1 usage   Khoa Tap Hat
{
    if(scenePhase == 0)
    {
        gp.bossBattleOn =true;

        //Shut the iron door to trap player
        for(int i = 0; i < gp.obj[1].length; i++) //Search a vacant slot for the iron door
        {
            if(gp.obj[gp.currentMap][i] == null) {...}
        }

/*...*/


        //Reset
        sceneNum = NA;
        scenePhase = 0;
        gp.gameState = gp.playState;

        //Change the music
        gp.stopMusic();
        gp.playMusic( i: 22);

    }
}
```

Figure 31: scene_skeletonLord function in CutsceneManager class

When a player kills the boss, the blocking objects will be removed by the checkDrop function of the MON_SkeletonLord class.

```
public void checkDrop() 1 usage  Khoa Tap Hat
{
    gp.bossBattleOn = false;
    Progress.skeletonLordDefeated = true;

    //Restore the previous music
    gp.stopMusic();
    gp.playMusic( i: 19);

    // Remove the iron doors
    for(int i = 0; i < gp.obj[1].length; i++)
    {
        if(gp.obj[gp.currentMap][i] != null && gp.obj[gp.currentMap][i].name.equals(OBJ_Door_Iron.objName))
        {
            gp.playSE( i: 21);
            gp.obj[gp.currentMap][i] = null;
        }
    }
```

Figure 32: checkDrop function in MON_SkeletonLord class

When the player finds OBJ_thay_nghia, the gameState changes to gp.gameState = gp.cutsceneState and calls the scene_ending function CutsceneManager to to execute the end game scene.

```
public class OBJ_thay_nghia extends Entity {  5 usages  Khoa Tap Hat *

    GamePanel gp;  5 usages
    public static final String objName = "Blue Heart";
    public OBJ_thay_nghia(GamePanel gp)  3 usages  Khoa Tap Hat
    {
        super(gp);

        this.gp = gp;

        type = type_pickupOnly;
        name = objName;
        down1 = setup( imagePath: "/objects/thay_nghia", gp.tileSize, gp.tileSize);
        setDialogues();
    }
    public void setDialogues()  1 usage  Khoa Tap Hat
    {
        dialogues[0][0] = "You found Teacher Nghia.";
        dialogues[0][1] = "You successfully rescued him from the dungeon.";
    }
    public boolean use(Entity entity) //when pickup this method will be called  Khoa Tap Hat *
    {
            gp.gameState = gp.cutsceneState;
        gp.csManager.sceneNum = gp.csManager.ending;
        return true;
    }

}
```

Figure 33: OBJ_thay_nghia class

```
public void scene_ending()  1 usage  Khoa Tap Hat
{...}
```

Figure 34: scene_ending function in CutsceneManager class

CHAPTER 3: ENTITY, NPCs, OTHER OBJECT DESIGN

This is one of the interesting chapters that you may read with enthusiasm because it contains lots of colorful figures. This chapter discusses the characters: the main character, enemies, NPCs, and finally other objects like chests, keys, and doors.

1. Player

Figure 35: Main Characters

The above characters are used for players during the game. With the purpose of diversifying characters and enhancing player experience, we have created 3 characters with 3 completely different formats so players can freely choose their favorite character.

2. Enemy

Figure 36: Enemy Characters

Monsters in video games serve several purposes. They provide challenges that keep players engaged and excited, offer variety in gameplay that requires strategic thinking, and showcase players' skills and strategic thinking.

For specific,

- BAT: has a high ability in movement that is difficult for player to defeat.
- MINIGROOT: is weakest monster in this game, but the amount of monster is a problem. Player need to handle strong boss while a lot of MINIGROOT is around.
- ORC: is one the the strongest monster with high damage, but restricted movement.
- ONEEYECANFLY: have widely damage that increases challenging and emotion of playing game.
- SKELETON LORD:the final boss with massive damage and dynamic movement. Player need to defeat the final to finish game and save Mr Nghia.

3. NPCs

Figure 37: NPCs

The NPCs above contain the characters along which pops up with dialogue when player interacts with the NPC. The dialogue system is simple just like other light RPG games.

Specifically:

- The Old Man is a character who provides guidance and direction to the player's character.
- The Merchant is a character who sells items, such as potions or other supplies, to the player.

4. Other Object

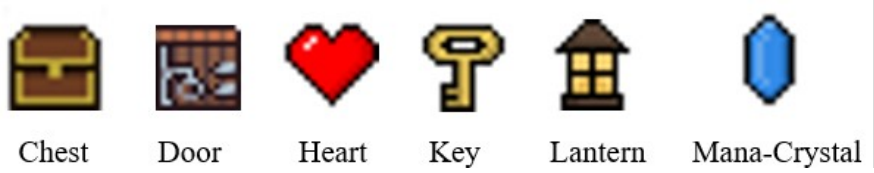

Figure 38: Other Object

The above objects do the following activities when player come in contract:

- Chest: Grants reward to the player whether it will be a coin, weapon,... the first time the player comes in contact.
- Door: Restricts players from entering other areas when they have not found the key.
- Key: To open door.
- Lantern: Helps players see in areas with limited light.
- Heart: Helps players to recovery their missing health.
- Mana-Crystal: Helps players to recovery their missing mana.

CHAPTER 4: MAP DESIGN

1. Asset

We have designed for dungeon for the game using the following asset below:

Figure 39: Map Tiles[*]

The provided text file "tiledata.txt" contains a list of image filenames paired with boolean values. Each "true" value indicates that the corresponding image represents a 2D box collider in our map design. The red boxes you see below are the 2D box colliders, which ensure that players and enemies stay within the game level.

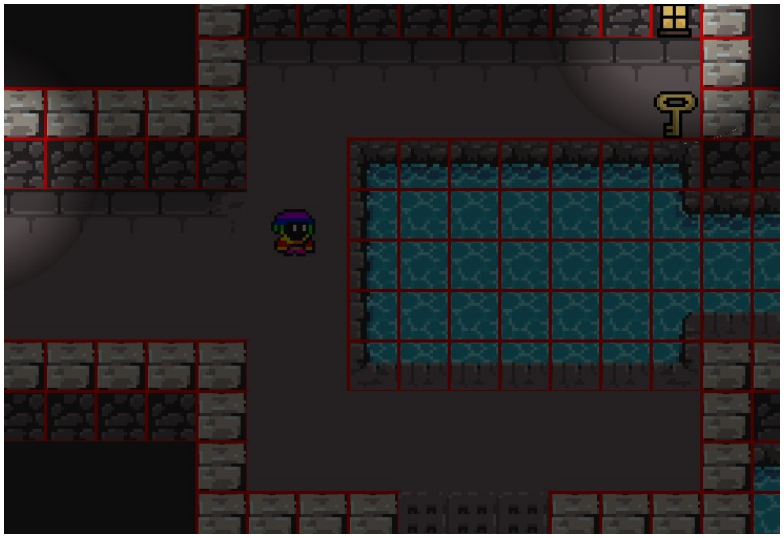

Figure 40: 2D Box Colliders

2. How To Design

We designed the map using Simple2DTileEditor 1.01 (EXE), a tool that lets you place and draw squares freely. It uses a technique of converting images into digital characters for storage and loading maps.

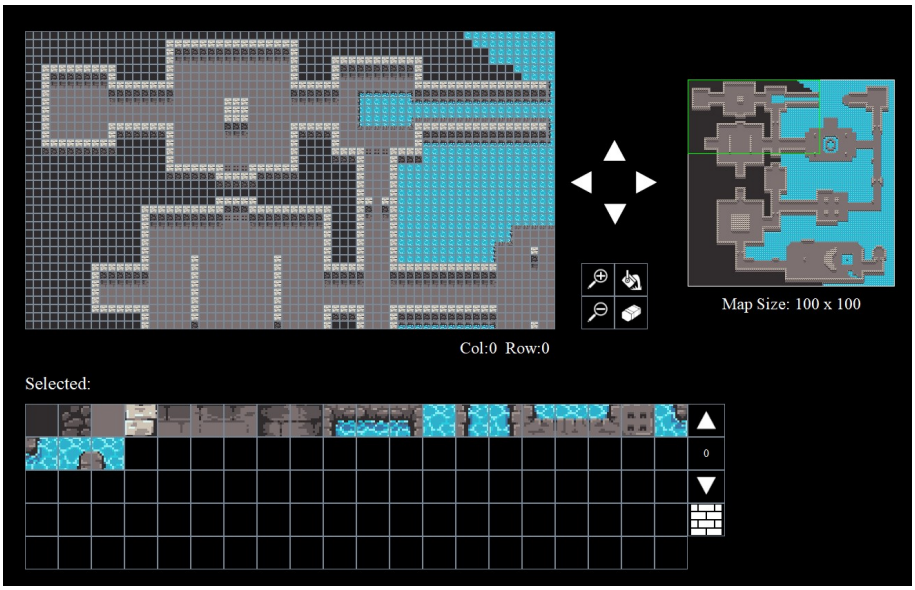

Figure 41: Overview Simple2DTileEditor 1.01

3. How To Load

This is described in the System Design Chapter.

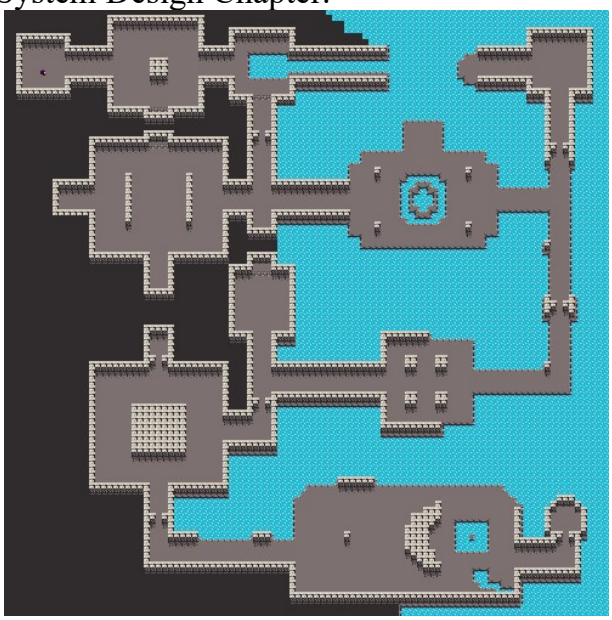

Figure 42: Overview Map 100x100

CHAPTER 5: GAME DESIGN

1. Game Rule

This game is a top-down dungeon game set within a single dungeon. Players control their character to find the key that unlocks the exit door, fight monsters like Bats, Orcs, and..., and ultimately defeat the powerful Boss Skeleton to achieve victory. Each area will contain different types of monsters, each with unique damage and health stats. Defeating monsters rewards players with money, experience, mana, and health restoration. Players can freely explore the dungeon, equipping weapons to enhance their combat abilities. They can also search for and use support items like health and mana restoration. Victory is achieved by defeating the Boss Skeleton and finding Sir. Nghia. However, if the player runs out of health, they lose the game.

2. Game Structure

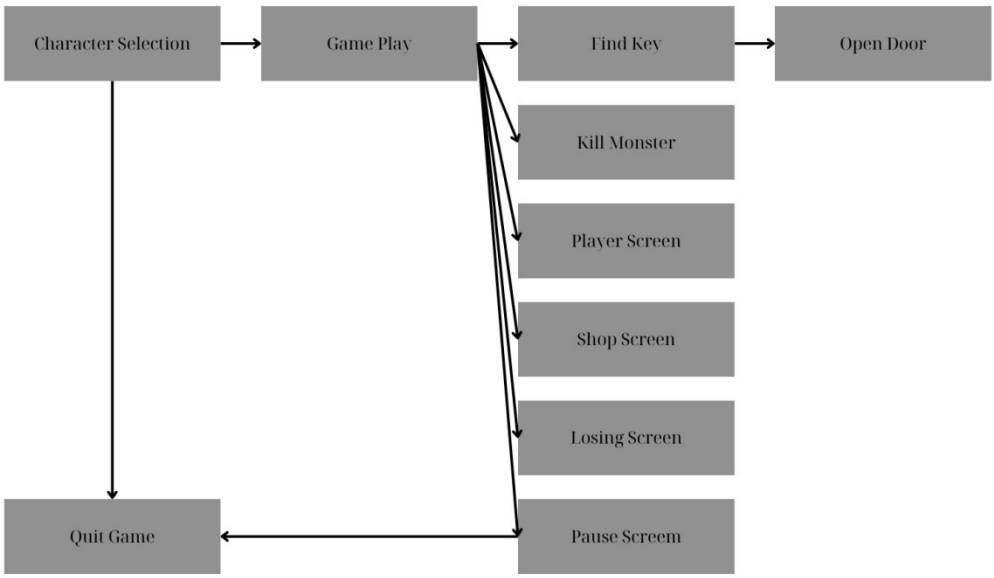

Figure 43: Game Structure

3. Game Play

Player Controls:

The following figure illustrates the input to control the player. We use keyboards key such as A,W,S,D to move the player and Enter to attack the enemies.

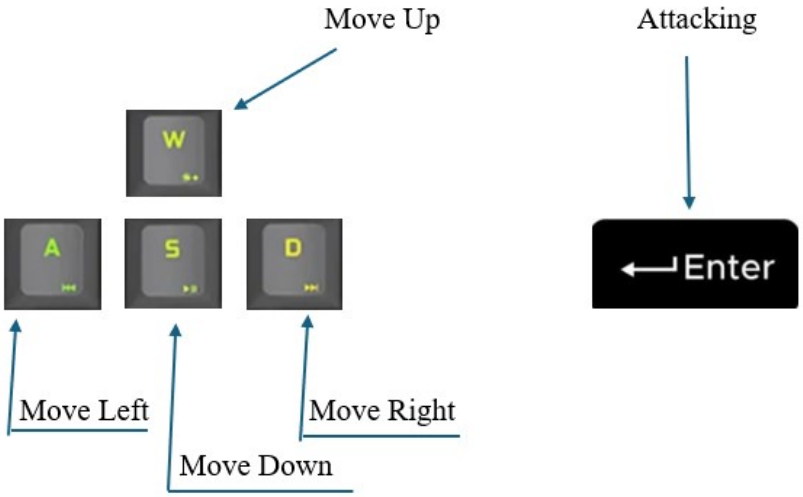

Figure 44: Player Controls

Game Controls:

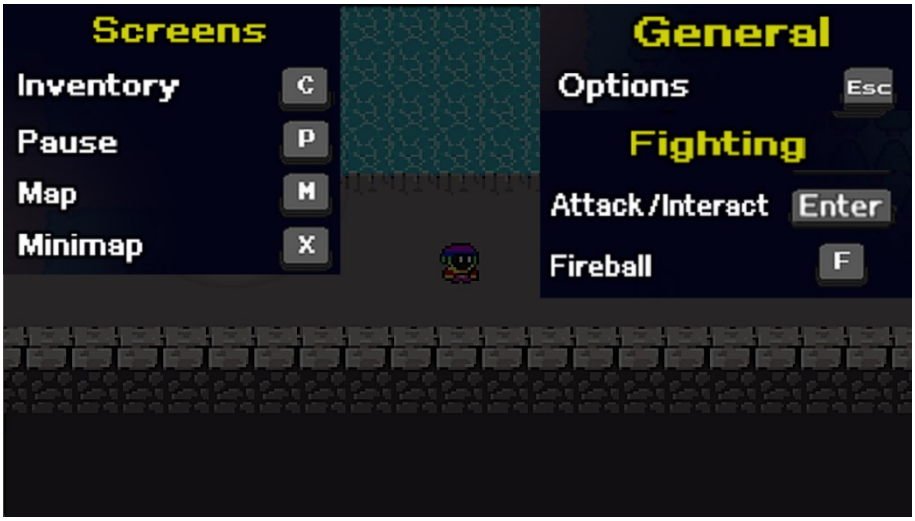

Figure 45: Game Controls

4. Animation States

4.1. Player:

The player has the total of 4 animation states: player_up, player_down, player_left, player_right, this is a 4-direction movement. We perform player movements by reading frames during each player move to create more realistic movement effects to help players experience the game as realistically as possible. Character orientation also will change based on the movement direction the player chooses. The following image illustrates the idea of directional movements.

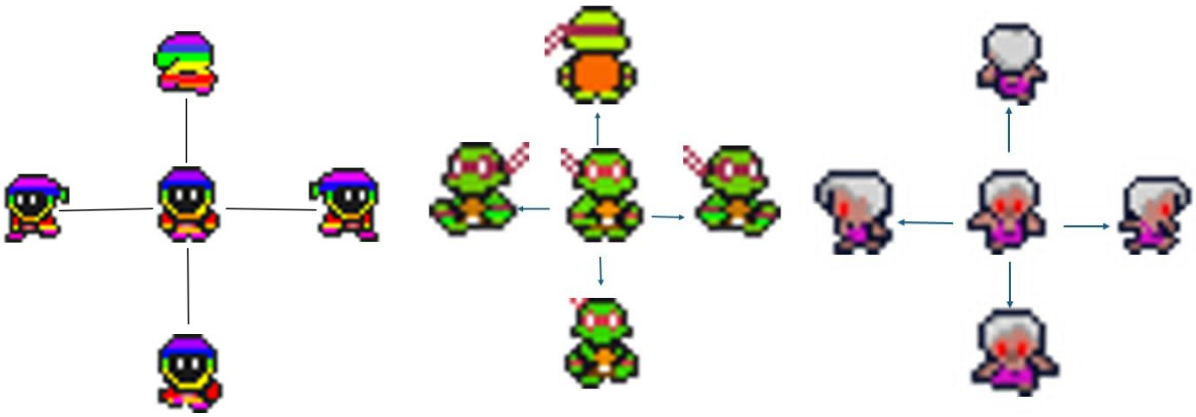

Figure 46: Player Animations

4.2. Player Attack:

For the player's attack, we also use frames reading for each attack, more specifically 2 images for 4 different directions (left, right, up and down) for every time the player attacks. The images below illustrates two of the player's four attacking direction, which is attacking left and right sides

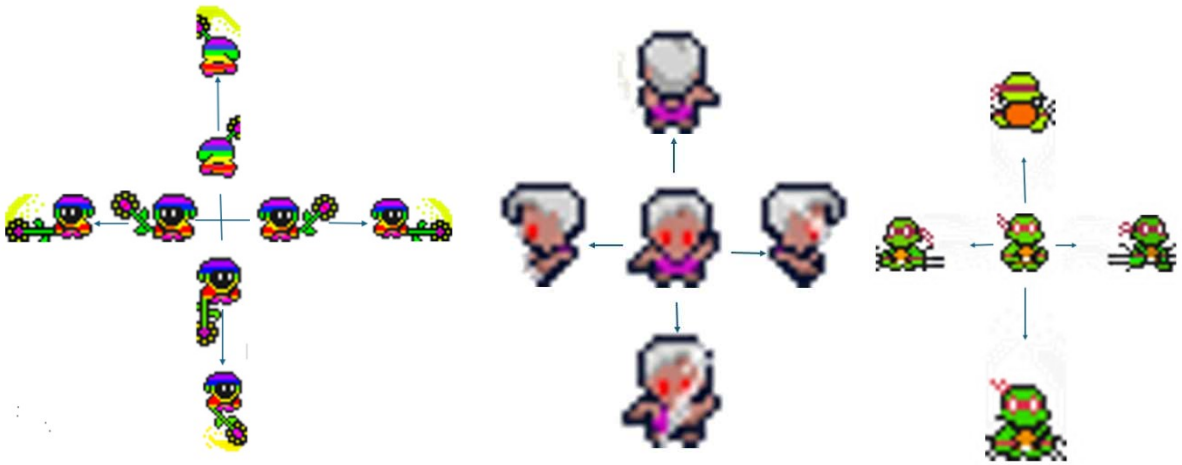

Figure 47: Player Attack

4.3. Monster:

The movement animation of NPCs and monsters is also built similarly to the player, by reading motion frames for moving directions up, down, left, and right.

4.4. Monster Attack:

For the monsters' attack, we use frames reading for each attack, more specifically 2 images for 4 different directions (left, right, up and down) for every time the player attacks.

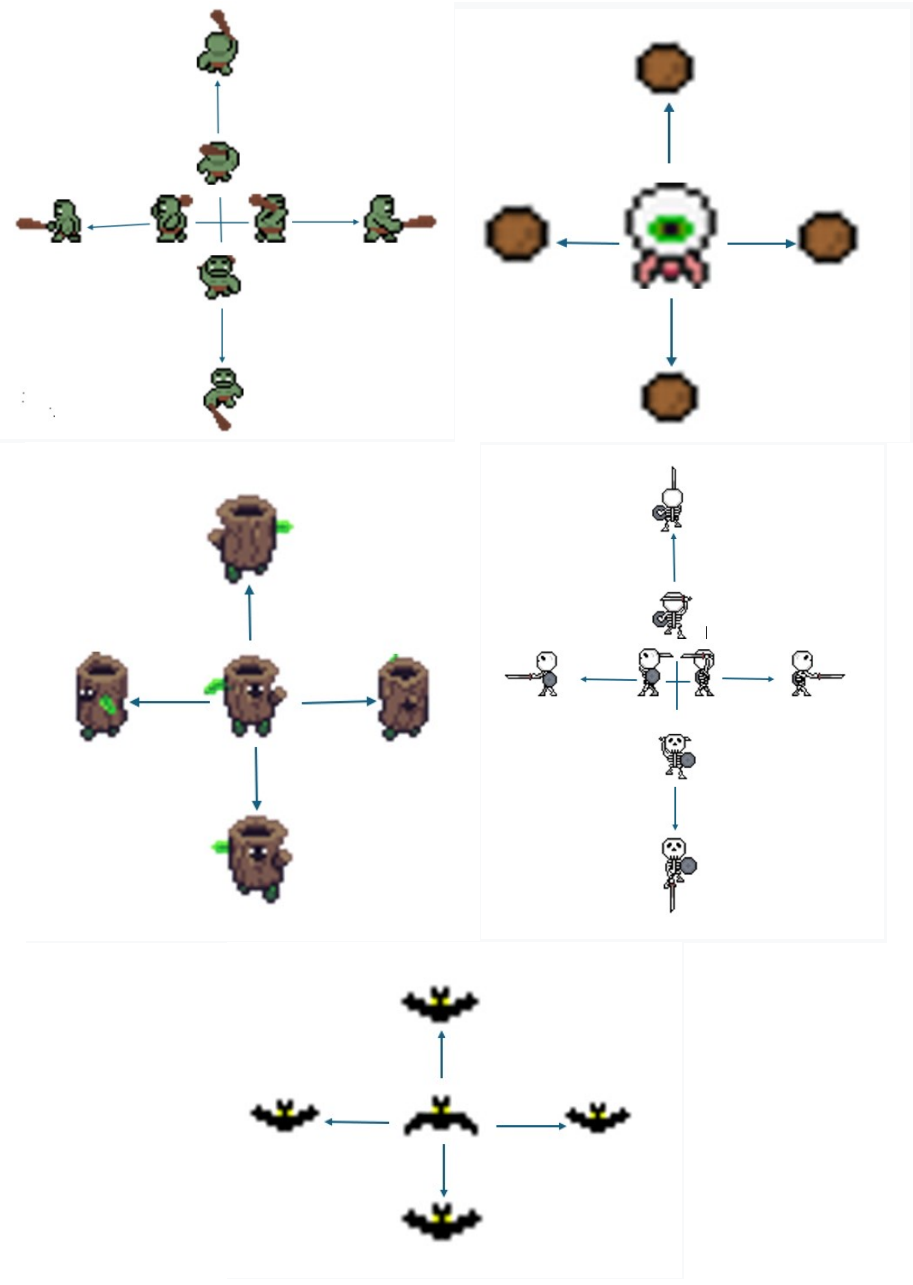

Figure 48: Monster Attack

5. UI

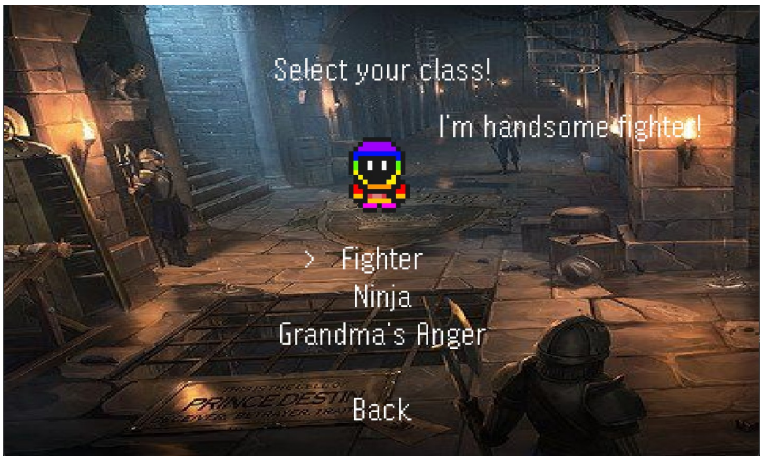

Figure 49: Select player

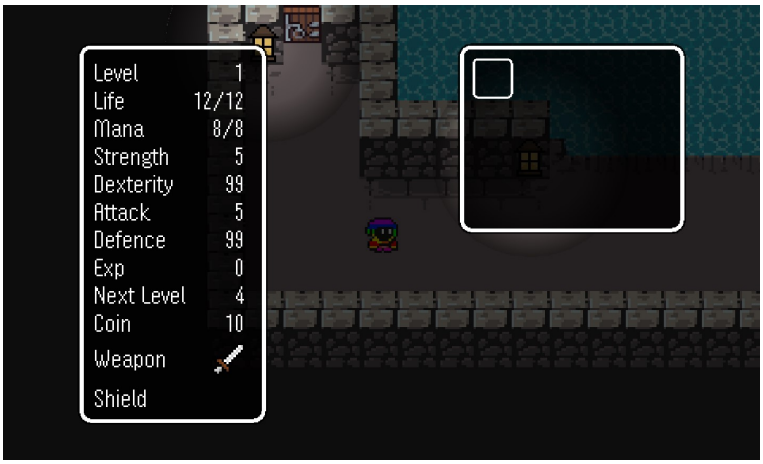

Figure 50: Inventory

Figure 51: Level up

Figure 52: Game pause

Figure 53: Game options

Figure 54: Game over

6. Audio

The following table shows the list of audio clips with descriptions our have used to develop this game:

Name	Function
levelup.wav	Notifies players every time the character levels up
hitmonster.wav	Uses every time the player attacks and damages monsters
blocked.wav	Notifies the player every time they move and hit an obstacle
BlueBoyAdventure.wav	Used throughout the game playing process, as the game's background music, preventing players from getting bored, increasing the drama of the game.
burning.wav	Used every time the player is burned by a monster's burning skill
coin.wav	Used every time player collecting coin
dooropen.wav	Used every time doors are opened
fanfare.wav	Used when the player completes all missions and wins the game
FinalBattle.wav	Used during the final fight
gameover.wav	Used when the player's HP reaches 0 and the character dies
powerup.wav	Used after the player has chosen to upgrade the character's power (mana, health, attack power)
speak.wav	Used every time the player interacts with NPCs
swingweapon.wav	Used as a sword swinging sound every time the player attacks

Table 2: List of audio

CHAPTER 6: DEMO

Click here to view: https://drive.google.com/file/d/1CLsoQxugP8uQHXMjLca8d3_UI-HAwY3w/view?usp=sharing

CHAPTER 7: CONCLUSION AND FUTURE WORKS

1. Conclusion

The development of the game is still ongoing. In the final phase, the team has gained a deeper understanding of the four core principles of Object-Oriented Programming (OOP) and the SOLID principles. This knowledge has significantly enhanced our proficiency in OOP for both game development and post-release programming, introducing novel features compared to the original version. The project classes have covered encapsulation extensively. Inheritance, abstraction, and polymorphism have been most frequently applied within the enemies' and player packages. Consequently, Saving Mrs Nghia was meticulously developed using the fundamental concepts of OOP, and the game code embodies all four key OOP features and a design pattern learned from class. The extensive knowledge gained from this experience is a testament not only to our collective expertise but also to the innovative spirit that has driven us to push the boundaries of game development.

2. Future works

For the timeline and restricted skills , we cannot do all plans that we have in mind. These features could enhance the player's abilities, special attack, and the new experiences for players including teleport, throwing firearms, and allowing players to customize their character with various skins, outfits, anh jewelries. In technical improvements, performance optimization will be updated to ensure smooth gameplay on various devices. This includes reducing load times, optimization asset usage, and improving frame rates. In content expansion, storyline enhancements will be committed, this would expand the game's narrative with additional storyline, quests, or side missions.

3. Acknowledgment

We would like to convey our deepest appreciation to our instructor and individuals who assisted us in reaching the goals of this project:

- Dr. Tran Thanh Tung and MSc. Nguyen Trung Nghia
- Original code from RyiSnow (https://youtu.be/om59cwR7psI?si=xngu7Kg2FalNV_4J)

REFERENCES

[1] BlueBoyAdvanture (2022).RyiSnow/blueboyadvanture[Java]. Retrieved May 10, 2024, from

https://github.com/berkayw/Blue-Boy-Adventure.git

[2] Java Swing Tutorial. Javatpoint. (n.d.). Retrieved December 11, 2023, from

https://www.javatpoint.com/java-swing

[3] UML Diagrams Tutorial. Javatpoint. (n.d.). Retrieved January 4, 2023, from

https://www.javatpoint.com/uml-diagrams

[4] Drew, O. (2022). Othneildrew/Best-README-Template. Retrieved January 4, 2023, from

https://github.com/othneildrew/Best-README-Template

[5] SOLID Principle in Programming: Understand With Real Life Examples. GeeksforGeeks. Retrieved December 28, 2023, from

https://www.geeksforgeeks.org/solid-principle-in-programming-understand-with-real-life-examples/

[*] aztharis.itch.io/tilemap-mini-dungeon, from

https://aztharis.itch.io/tilemap-mini-dungeon