VIETNAM NATIONAL UNIVERSITY – HCM

INTERNATIONAL UNIVERSITY

SEMESTER 1 (2024-2025)

OBJECT - ORIENTED PROGRAMMING

CHRISTMAS TREASURE GAME

Members:
Lê Phan Hồng Huế - ITDSIU23032
Lê Thị Ngọc Tuyền - ITDSIU23028
Nguyễn Thị Diễm Quỳnh - ITDSIU23021
Trần Thị Kiều My - ITCSIU23026

TABLE OF CONTENT

CHAPTER 1: INTRODUCTION 4
CHAPTER 2: RULE AND GAME PLAY 4
CHAPTER 3: THE LOGIC DETAILS AND GAME TECHNIQUE 17
1. KeyHandler.java: 17
2. Entity.java: 18
3. Player.java: 19
4. GamePanel.java: 20
5. CollisionChecker.java: 21
6. AssetSetter.java: 21
CHAPTER 4: UML CLASS DIAGRAM 22
CHAPTER 5: EVALUATION 28
1. Performance Evaluation: 28
2. Code Quality 29
CHAPTER 6: EXPERIENCE 29

LIST OF FIGURES

Figure 1: Main menu 5
Figure 2: Maze landscape 6
Figure 3: Game Items 7
Figure 4: Heads-Up Display and Inventory Screen 7
Figure 5: Use Potion_Red 8
Figure 6: Attack Monster 9
Figure 7: Snowball Weapon 9
Figure 8: Healing Pool 10
Figure 9: Character is attacked 11
Figure 10: Game Over Screen 11
Figure 11: Options Screen 12
Figure 12: A Pit 13
Figure 13: Level Up 13
Figure 14: Random Items after kill SnowBall Monster 14
Figure 15: Chat with NPCs 14
Figure 16: Biscuit Monster 15
Figure 17: A Treasure 16
Figure 18: Winning Game 17

CHAPTER 1: INTRODUCTION

For our final project in the Object-Oriented Programming (OOP) course, our professor asked us to create a game that demonstrates the key principles of OOP. We decided to develop a 2D platformer game called "Christmas Treasure" showcasing OOP features like inheritance, abstraction, encapsulation, and polymorphism, as well as design patterns such as the Singleton pattern.

We used Java as the programming language for this project, as it is an object-oriented language that allows us to effectively apply OOP concepts in our code.

This report will provide a detailed overview of the game, including its creation process, gameplay mechanics, and how players interact with it. Additionally, we will discuss how we applied OOP principles in the development of the game.

CHAPTER 2: RULE AND GAME PLAY

The game we created is a **top-down 2D platformer** inspired by a youtuber named RyoSnow. The player takes on the role of **Black Knight**, a lost knight trapped in a mysterious maze filled with Christmas trees. To escape, Black Knight must fight Snowman monsters along the way and follow the guidance of **Old Man** to find the hidden **treasure left by Santa Claus** within the maze.

To play this game, we always use four main push buttons: **W** to go up, **A** to turn left, **S** to move down, and **D** to turn right.Another push button is **ENTER**, which is primarily used to attack snowman monsters with weapons such as swords, axes. Additionally, it is used to select items from the inventory and to interact with NPCs created within the game.Moreover , there are other functional buttons such as **P**, which pauses the game; **T**, to view the current position of the character on the map; **C**, to open the inventory and check the character's status; **Space**, to shoot snowballs; and **ESC**, which allows players to adjust the volume of music and sound effects or exit the game.

"Christmas Treasure" game has one maze map, and in this report, we will explain everything inside the game and guide you on how to play and win.

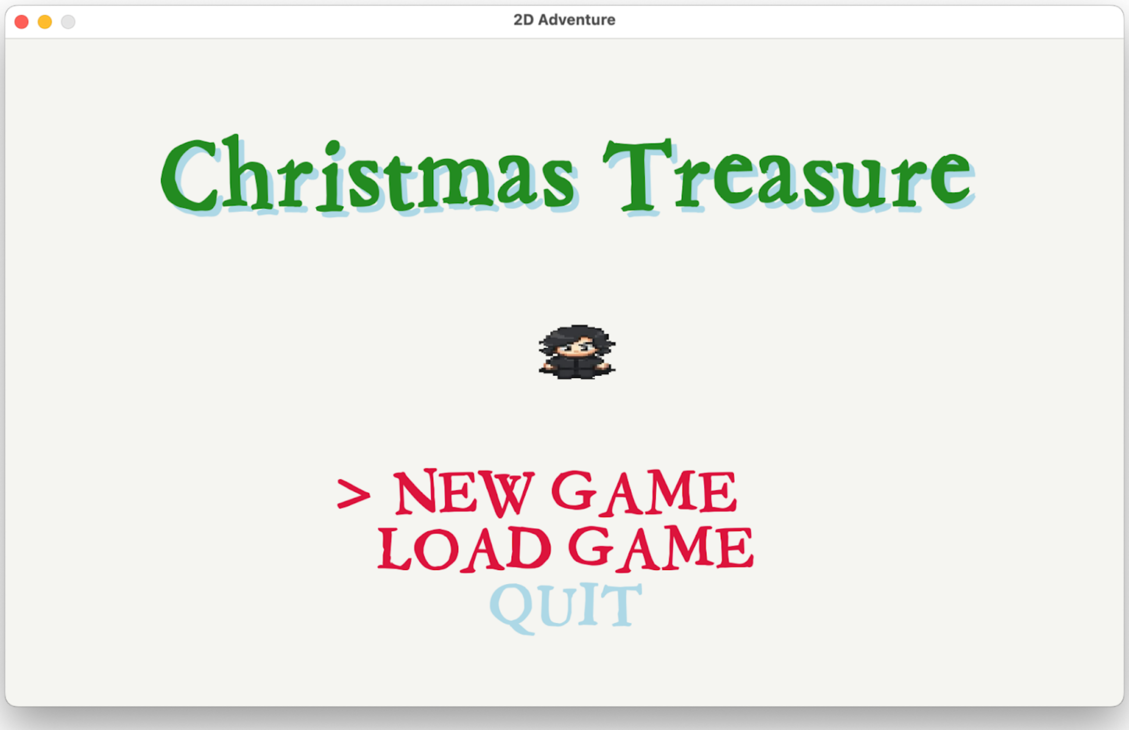


Figure 1: Main menu

Click "New Game" to embark on your adventure, or select "Quit" to exit the application. Starting a new game will transport you to the maze map, the heart of *Christmas Treasure*. Here, your journey unfolds as you navigate through intricate pathways, facing diverse challenges and obstacles. This map sets the stage for your adventure, introducing the captivating environment, formidable enemies, and engaging gameplay mechanics that drive the story and immerse you in the quest to uncover Santa's hidden treasure.

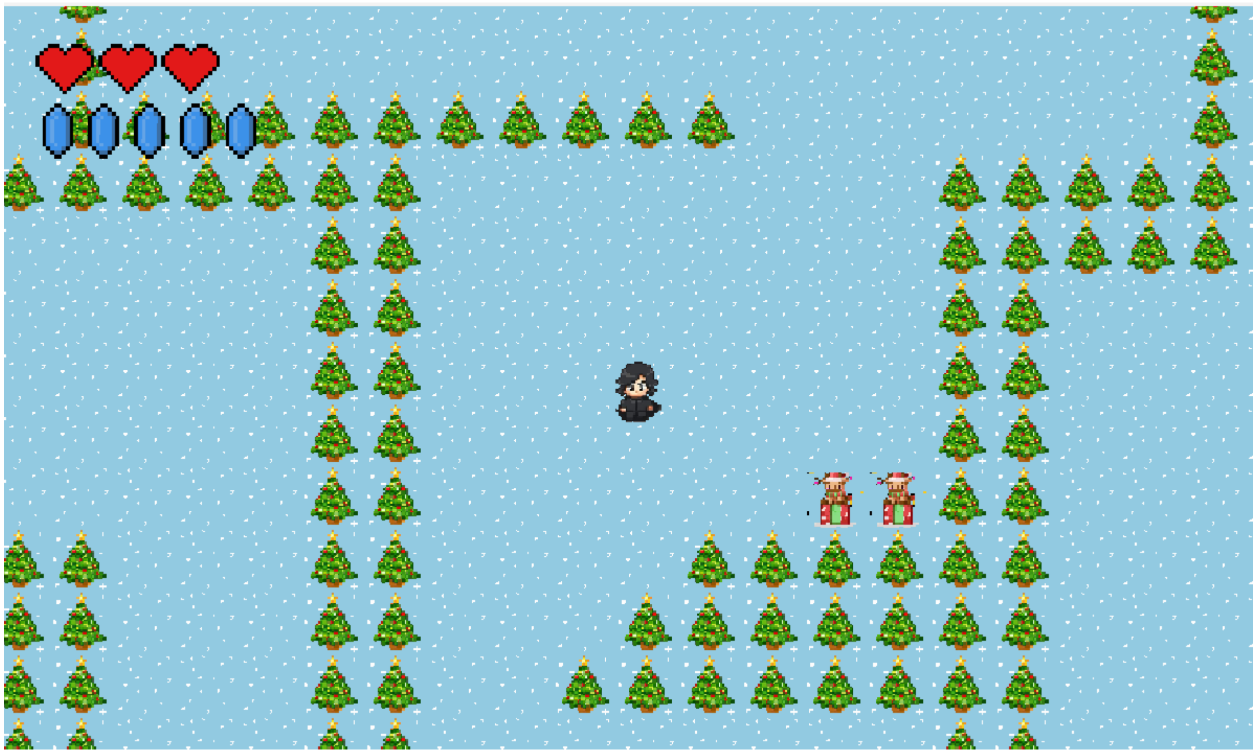


Figure 2: Maze landscape

In this maze, the player will control the character to move around the map, collecting items and finding a way to unlock the treasure left by Santa Claus in order to escape. However, to collect items, the player must face monsters called Snowmen that appear on the map. When defeated, these monsters may randomly drop items such as mana or health potions to restore energy. Additionally, items can also be found scattered along the path, providing further opportunities for the player to gather and equip for the journey.

Furthermore, defeating monsters will help players gain experience and level up, making it easier to defeat monsters in the future.

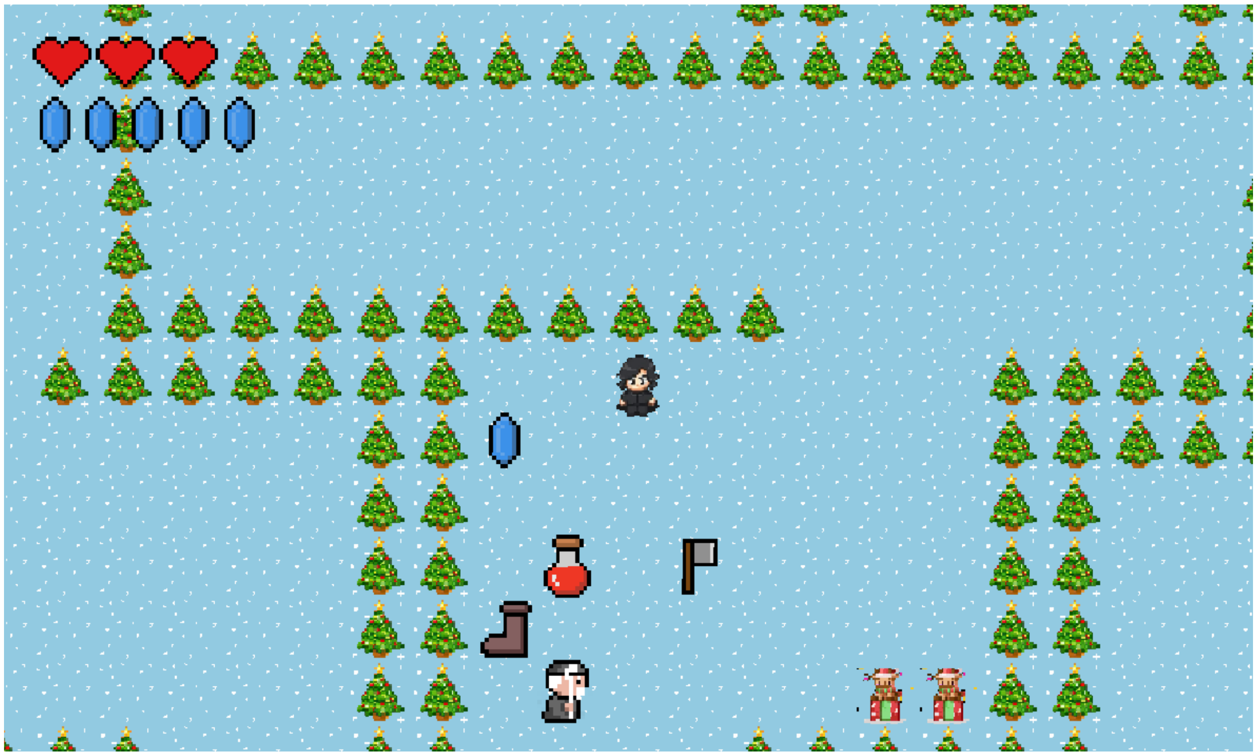
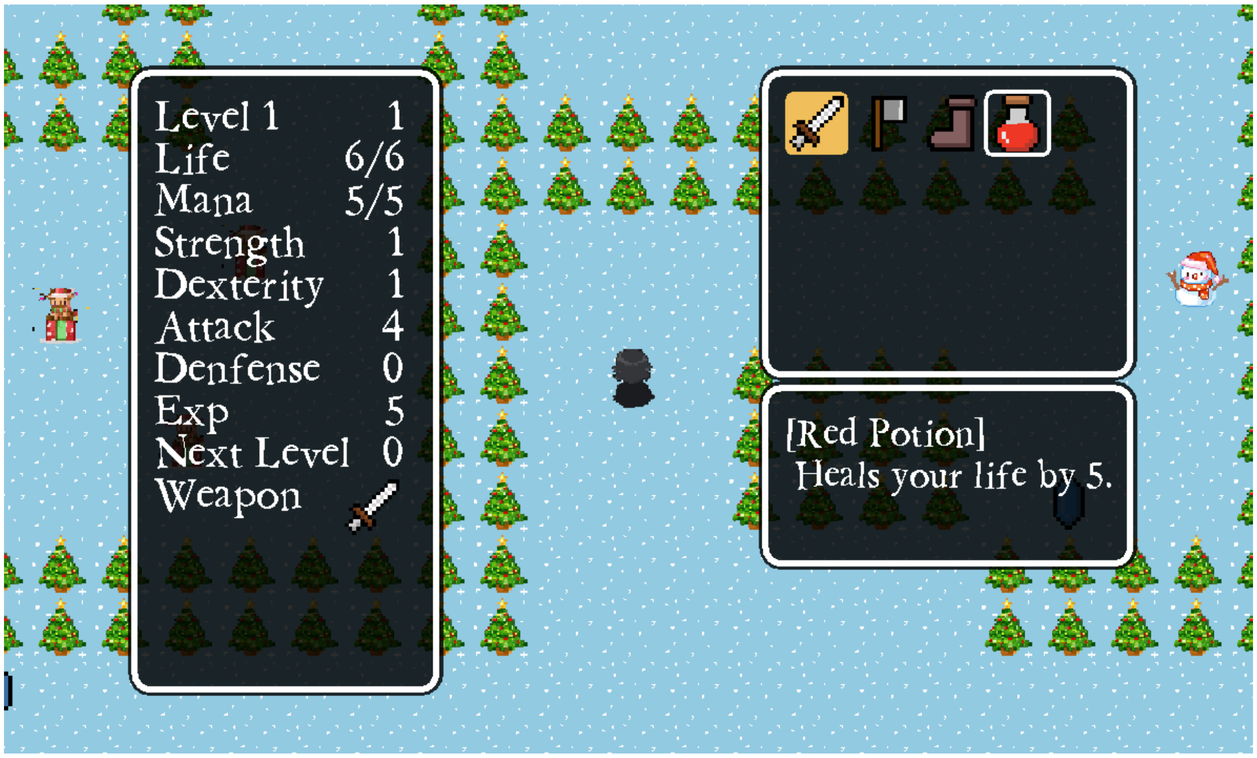


Figure 3: Game Items



```
Level 1       1
Life        6/6
Mana        5/5
Strength      1
Dexterity     1
Attack        4
Denfense      0
Exp           5
Next Level    0
Weapon
```

[Red Potion]
Heals your life by 5.

Figure 4: Heads-Up Display and Inventory Screen

There are several items that are created for users to obtain and make it simpler to cut trees, kill the snowman .They are red_potion, axe, sword_normal, chirstmas_boot ,manacrystal,...At the same time, it also shows other parameters such as the player's experience,level or the weapon being used,... to make it easier to observe

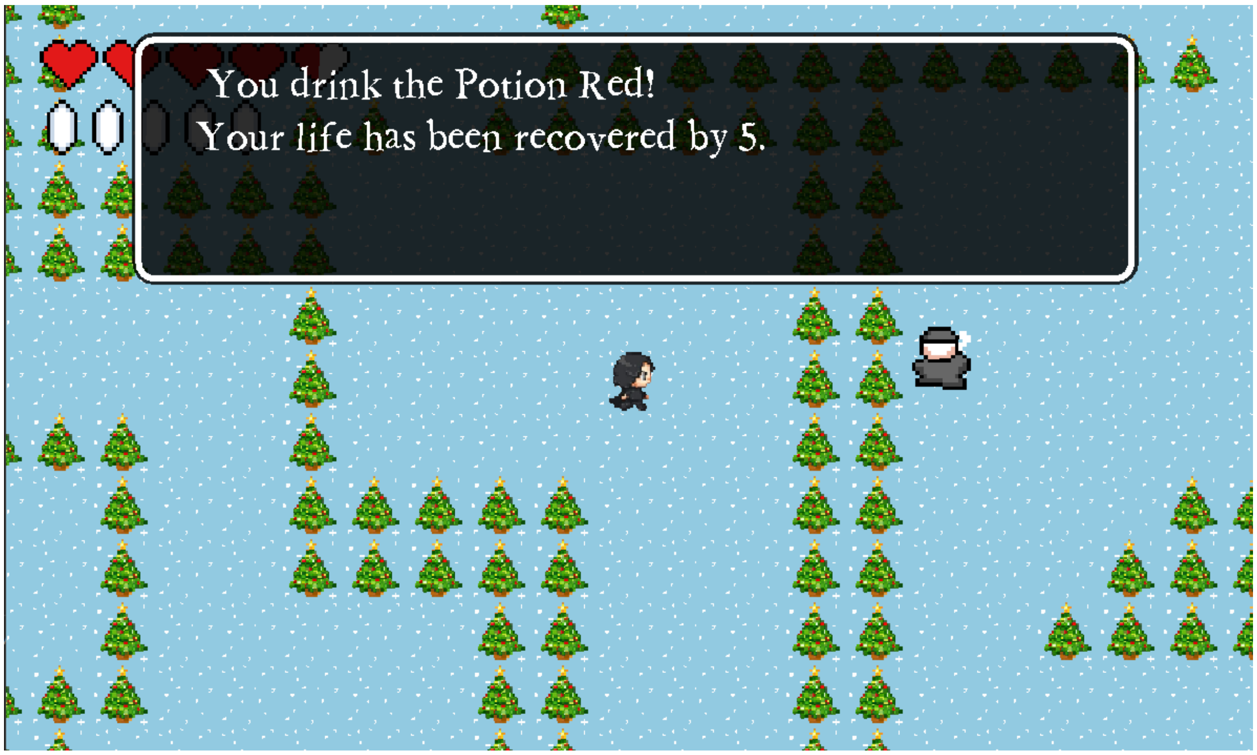

You drink the Potion Red!
Your life has been recovered by 5.

Figure 5: Use Potion_Red

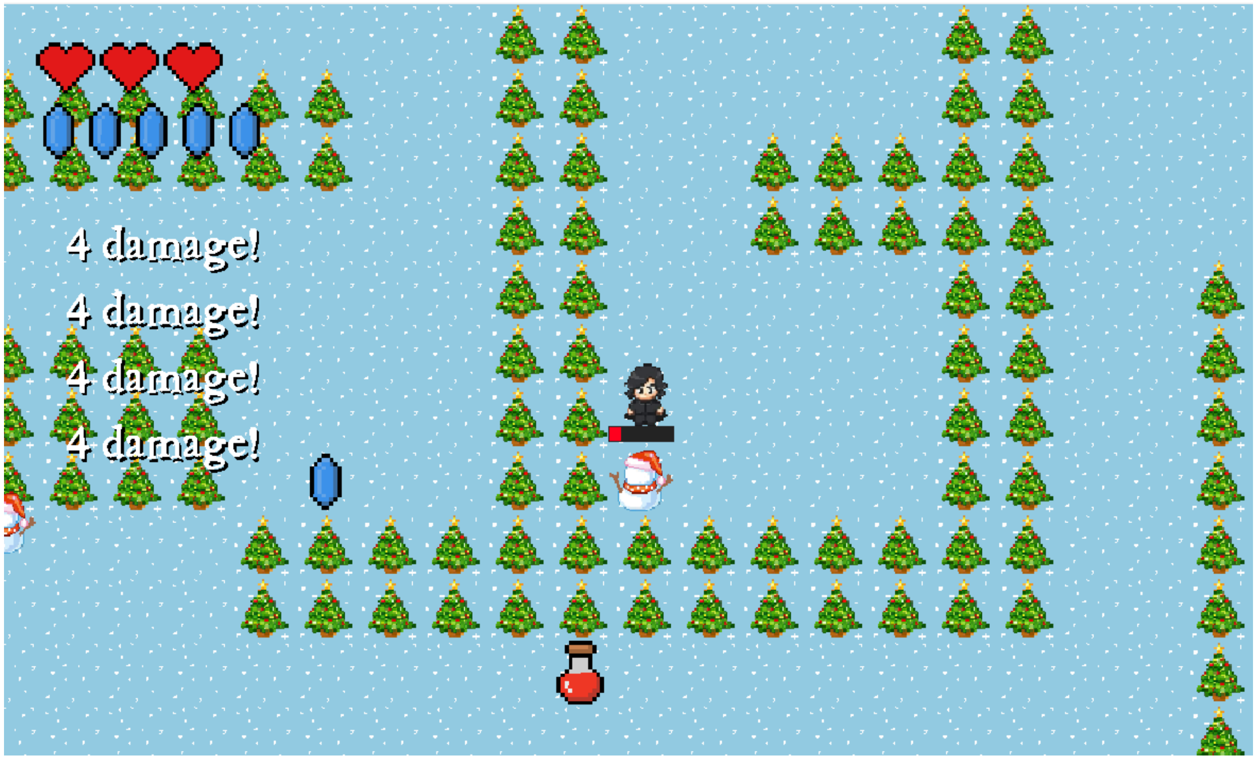


Figure 6: Attack Monster

Each character's attack will cause a certain amount of damage to the Snowman depending on the type of weapon used.

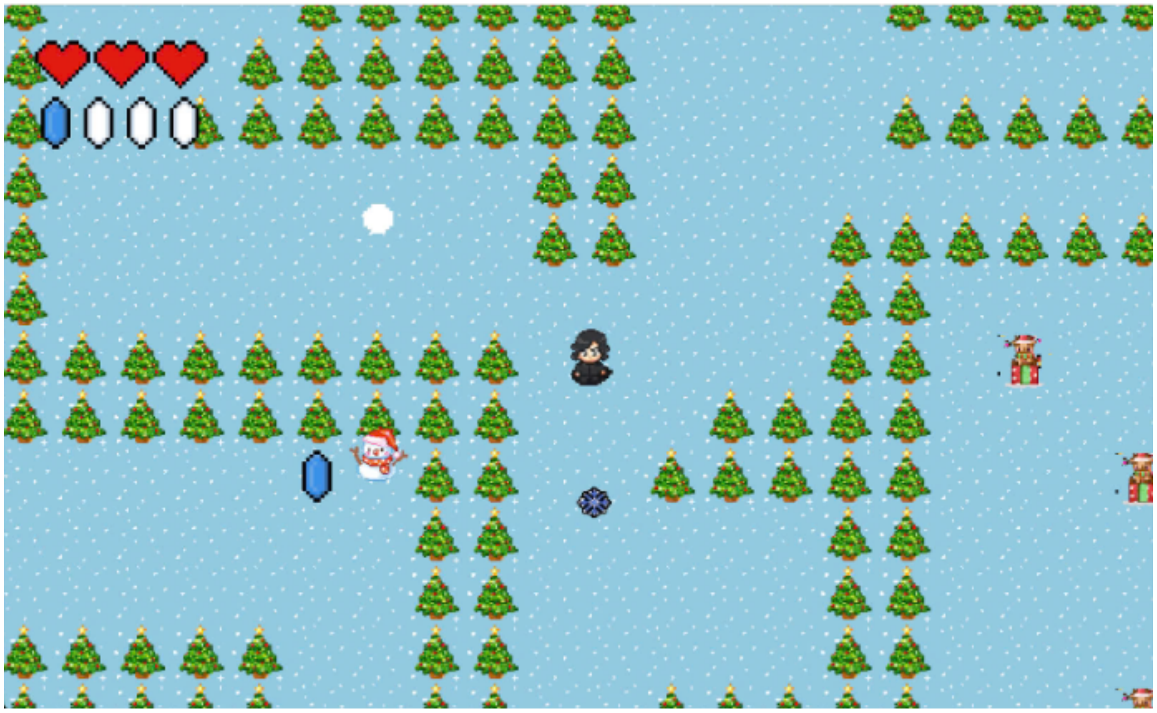


Figure 7: Snowball Weapon

This amount of mana is used to launch snowballs at monsters. When you run out of mana, it cannot be used. This is also the weapon with the most powerful damage. Mana can be restored by gathering it along the way.

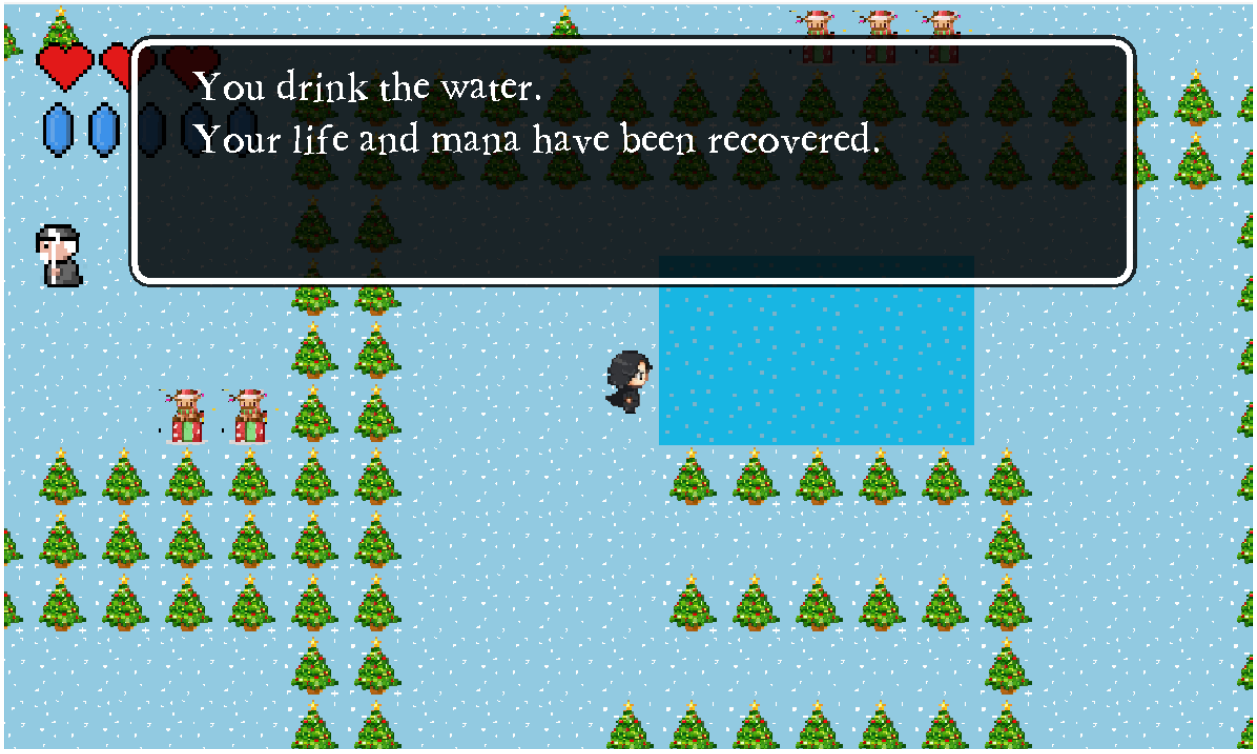

You drink the water.
Your life and mana have been recovered.

Figure 8: Healing Pool

In addition, this game also supports players in recovering mana with healing pools

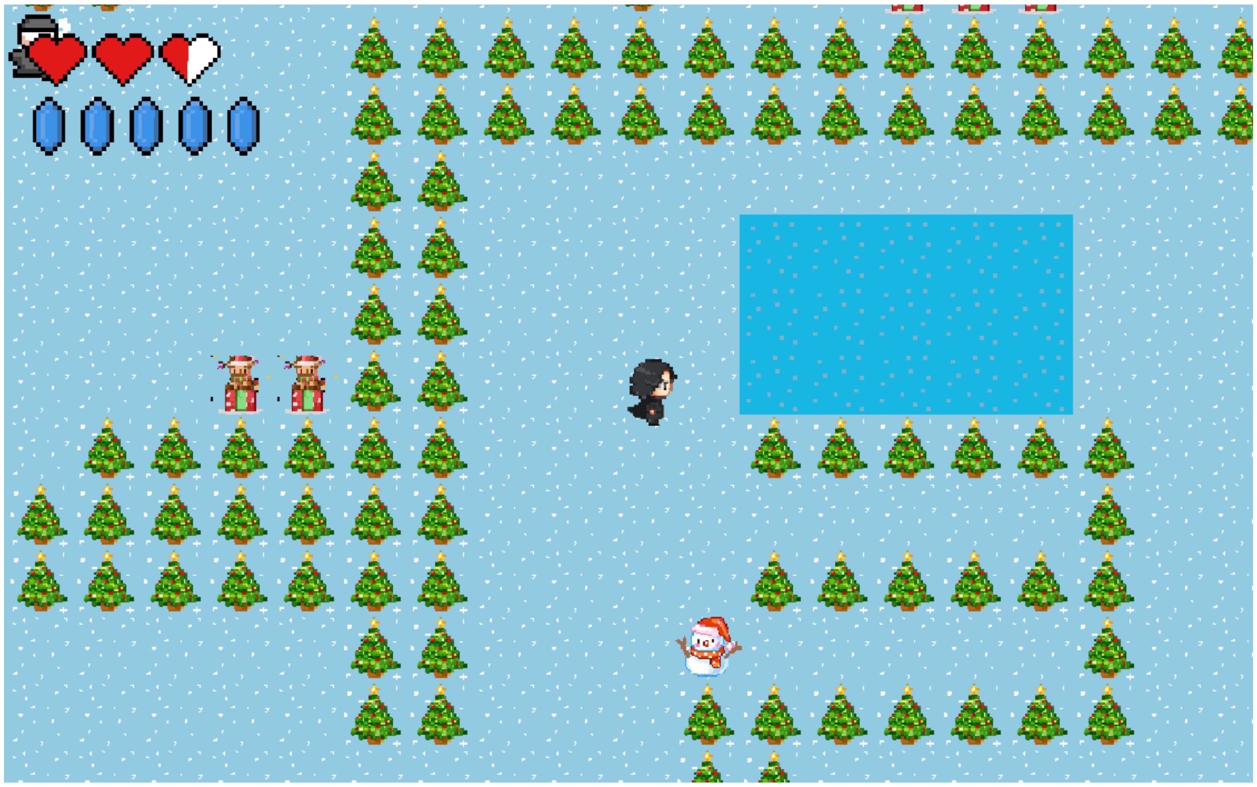


Figure 9: Character is attacked

Each time the character is attacked by Snowmans, he will lose half of his heart. If he loses all his heart, he will lose

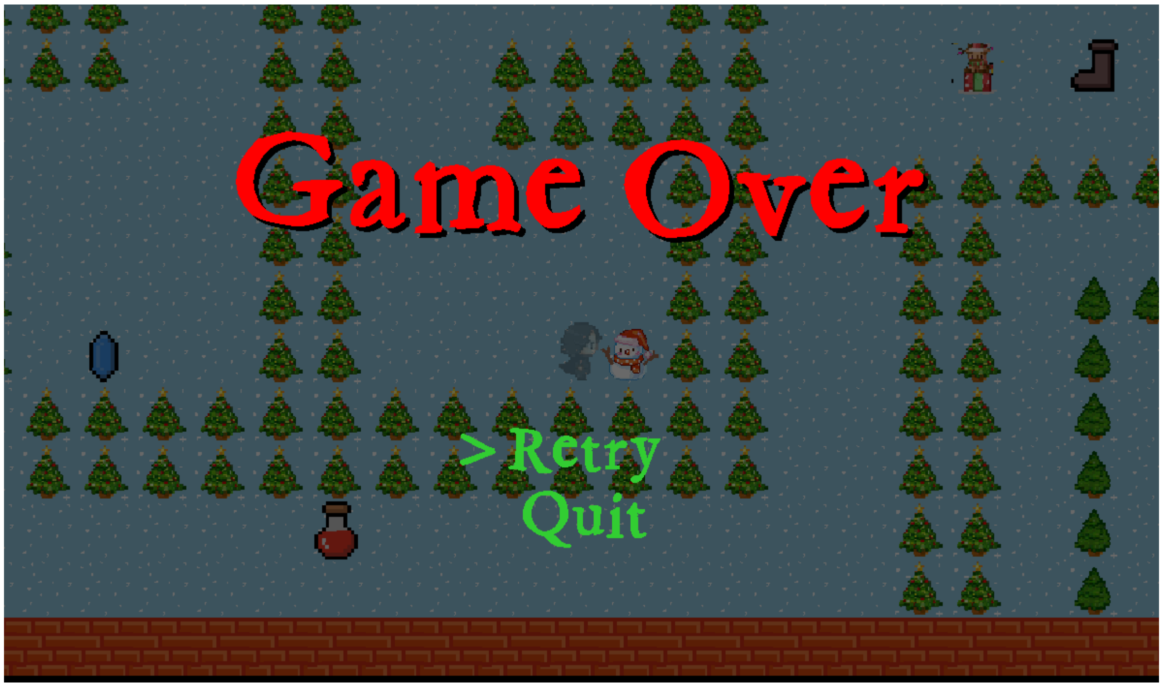


Figure 10: Game Over Screen

After losing, you have two options, one is "RETRY" to be revived at the original position when entering the game with all the items and levels available before being defeated. The other is "QUIT" will help you exit the game completely and return to the NEW GAME lobby.

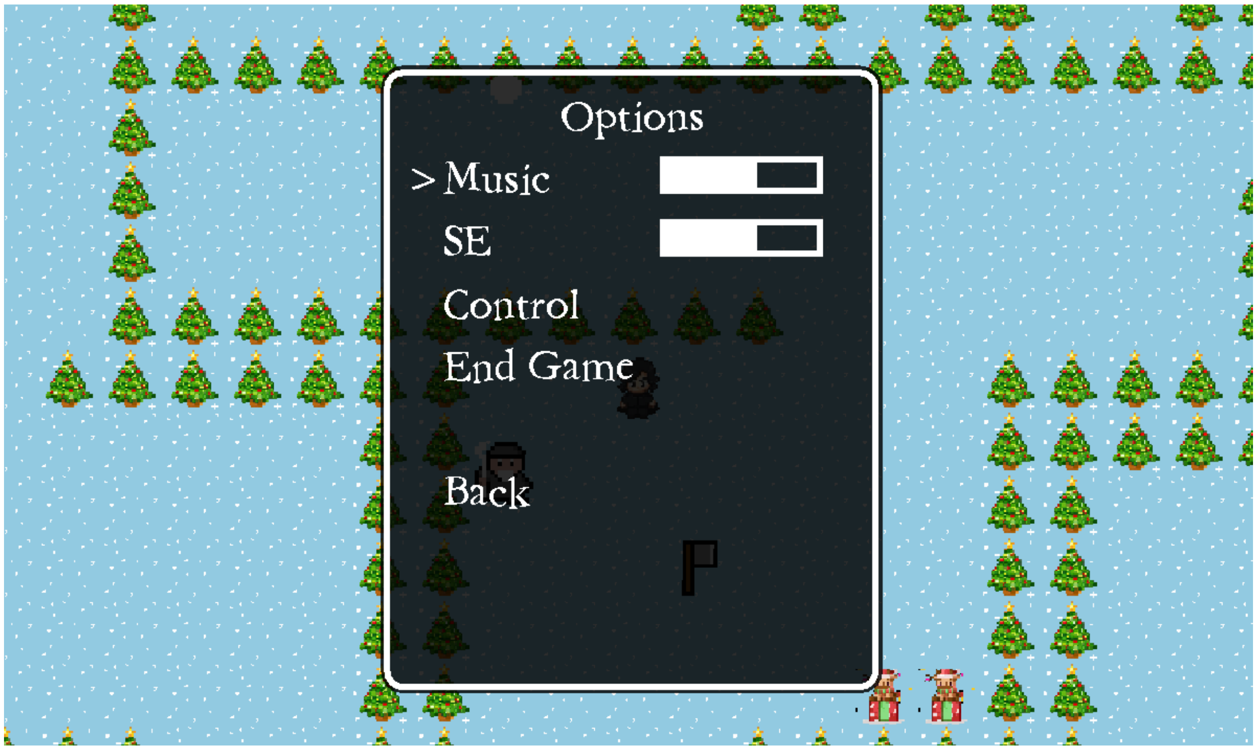


Figure 11: Options Screen

Moreover, in the game the ESC button can also adjust the volume and if the player wants to end the game immediately, they can select "END GAME" and return to the NEW GAME lobby without having to be defeated by monsters. The Control can help the player understand each button link with the keyboard.

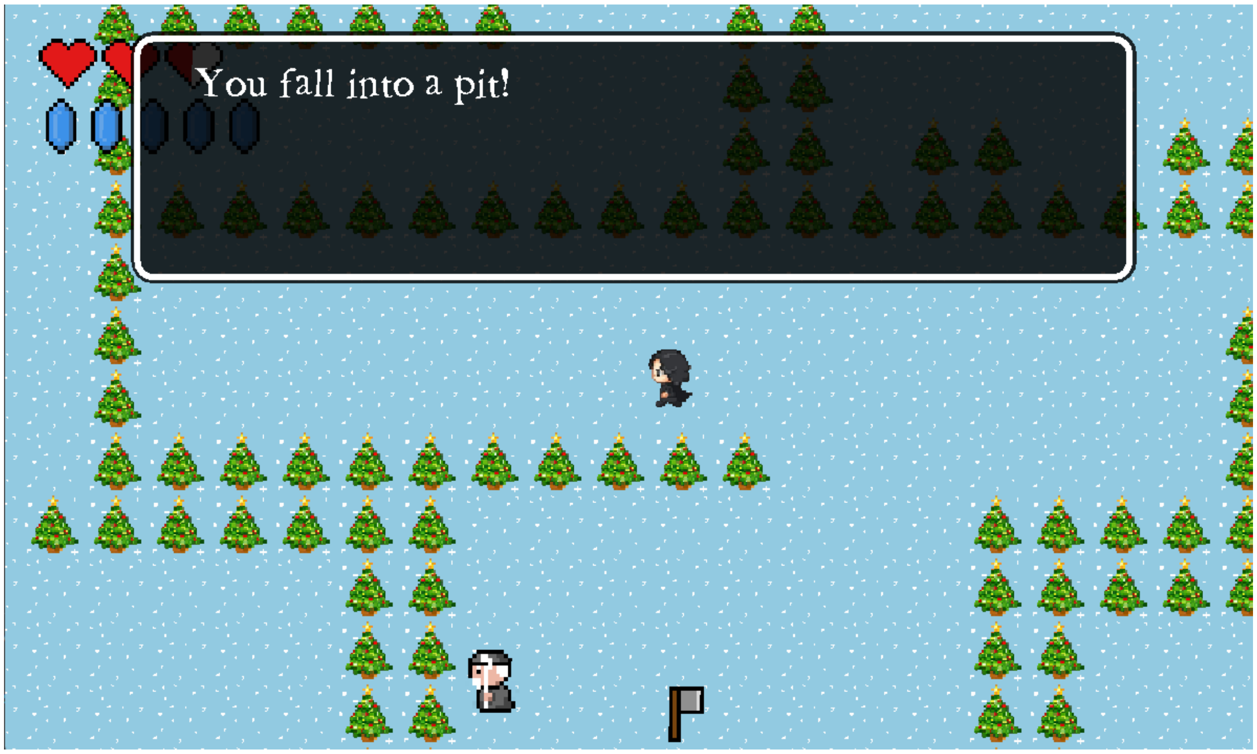

You fall into a pit!

Figure 12: A Pit

Besides Snowmans, traps are also dangerous things that need to be avoided if you don't want to lose your life

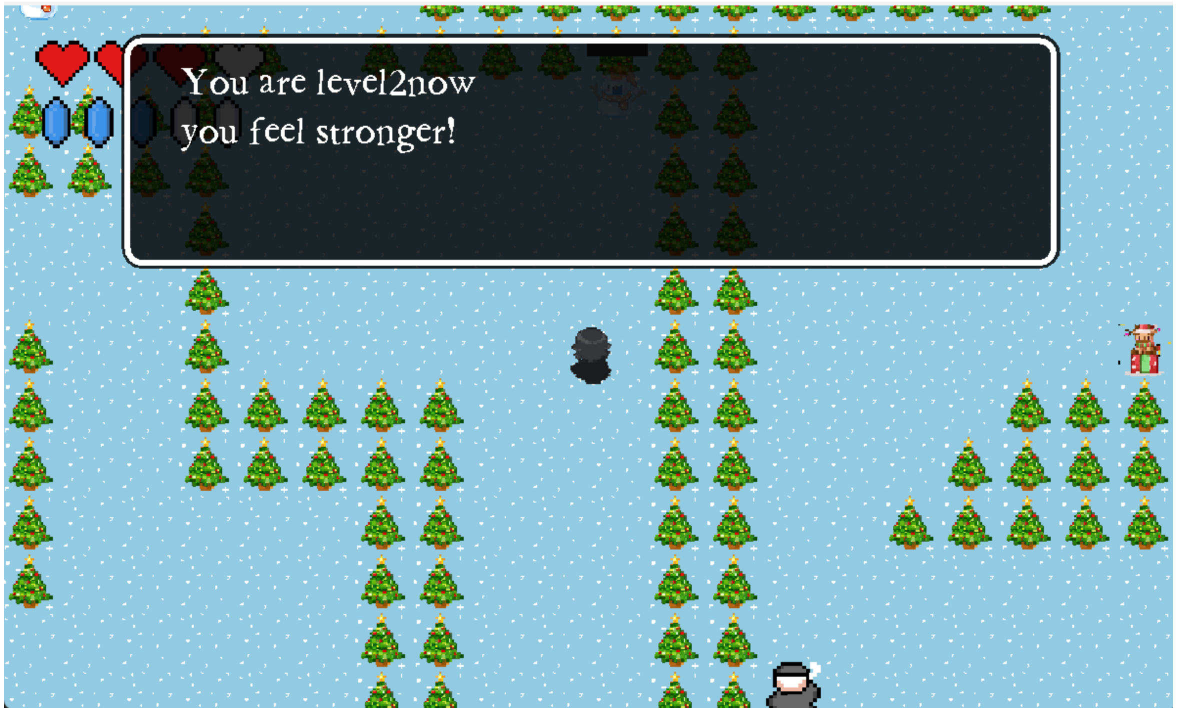

You are level2now
you feel stronger!

Figure 13: Level Up

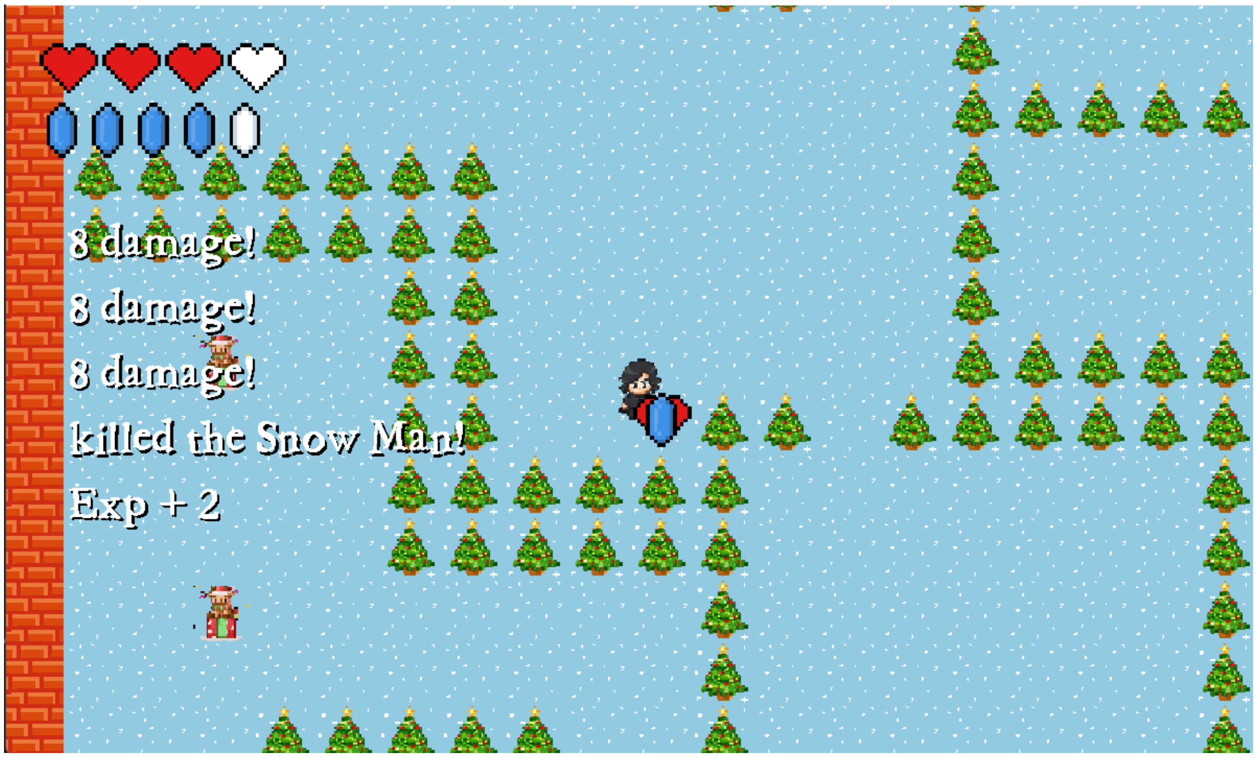


Figure 14: Random Items after kill SnowBall Monster

Killing more monsters will help you gain more experience and level up, while also increasing the amount of damage you create significantly.

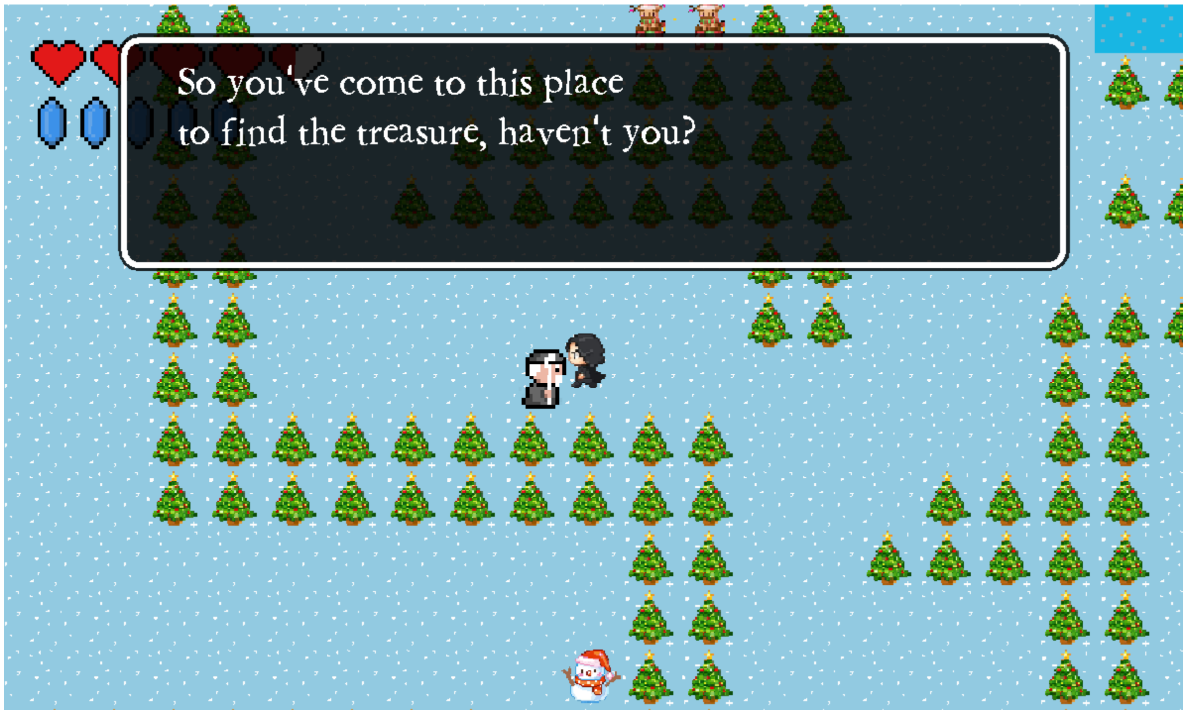

So you've come to this place
to find the treasure, haven't you?

Figure 15: Chat with NPCs

You can chat with NPCs to understand the meaning and how to win the adventure.

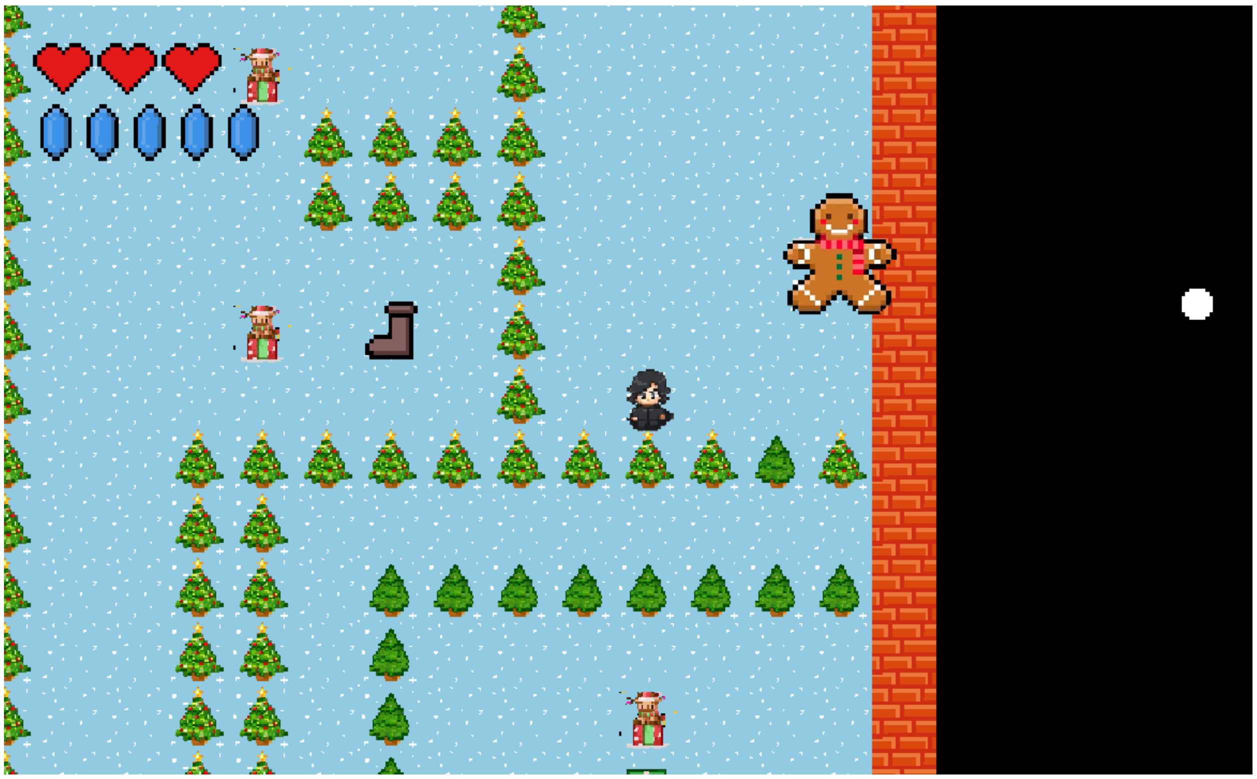


Figure 16: Biscuit Monster

The final treasure will be guarded by a dangerous Biscuit monster. If you want to find the treasure, you must defeat it

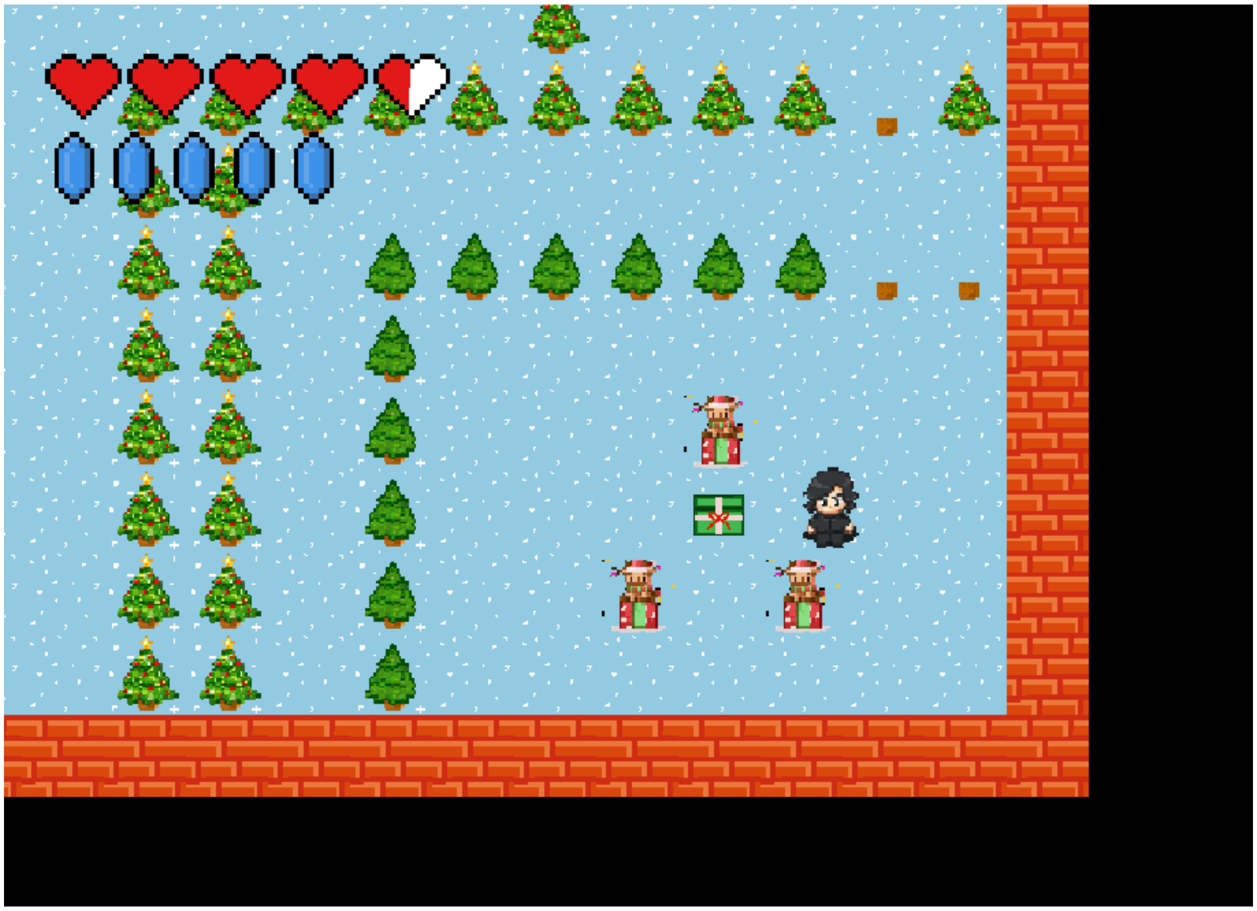


Figure 17: A Treasure

This is the gift that Santa left, open it to win the game.

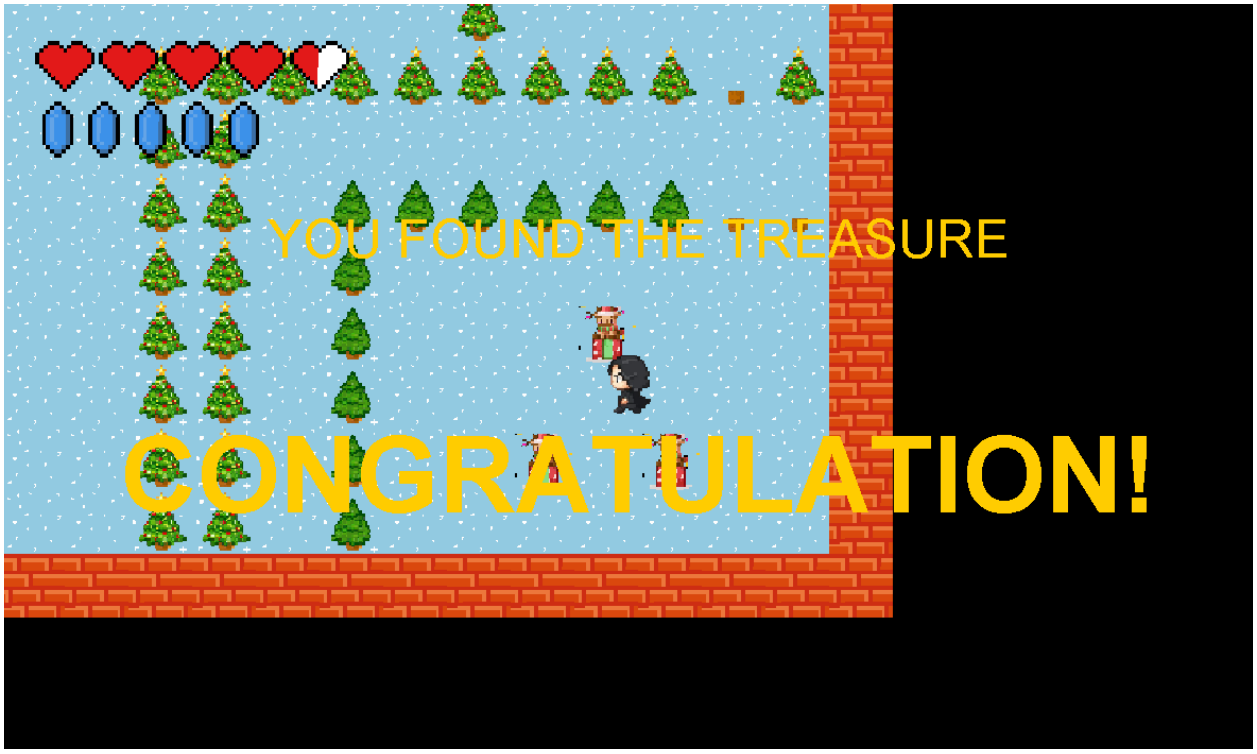


Figure 18: Winning Game

CHAPTER 3: THE LOGIC DETAILS AND GAME TECHNIQUE

1. KeyHandler.java:

Keyboard Input Handling:
The game responds to key presses and releases, translating them into specific in-game actions. It also manages multiple game states, such as the title screen, gameplay, pause menu, dialogue, character interactions, options menu, game over screen, and trade interface. Each state is triggered by relevant keyboard inputs, allowing the player to navigate seamlessly between different aspects of the game.

Game State Transitions:
The system facilitates seamless transitions between various game states in response to user input. Players can move between menu screens, initiate dialogues, and control character movements, ensuring a smooth and intuitive gameplay experience

Inventory Interaction:
The system allows players to interact with their inventory during the character state, enabling them to navigate through inventory slots and select items. It also manages inventory interactions during trade sequences, facilitating item exchanges between the player and non-player characters (NPC).

Debug Features:
The game includes debug tools that allow developers to toggle the visibility of debug text and other diagnostic elements, helping to identify and resolve issues during development.

Volume Adjustment and Options Menu:
The system handles user input for adjusting volume levels within the options menu. It also manages navigation in the menu, enabling players to customize various game settings, such as sound, graphics, and controls.

Event Triggers:
The game listens for specific keys to activate in-game triggers, such as starting dialogues, executing character actions, or pausing the game. These triggers ensure responsive and dynamic gameplay interactions

2. Entity.java:

Position and Collision Handling:

Manages entity positions (using worldX and worldY) and collision detection for interactions with other entities, objects, and obstacles. Utilizes solidArea and collisionOn to handle movement restrictions.

Sprite Animation:

Animates entity movements through sprite sheets by alternating images (spriteNum) and controlling transitions using a spriteCounter. Animations are direction-based (up, down, left, right).

Attributes and Counters:

Defines attributes such as speed, life, mana, and level. Uses counters for actions (actionLockCounter), invincibility (invincibleCounter), shooting availability (shotAvailableCounter), and dying animations (dyingCounter).

Entity Types:

Categorizes entities into types (e.g., player, NPC, monster) using constants like type_player, type_npc, and type_monster, allowing easy identification and classification.

Inventory and Items:

Manages inventory items with attributes such as value, attackValue, defenseValue, and description. Supports item usage with the use method and item drops using dropItem.

Interactions and Dialogue:

Enables interactions between entities, including speaking (speak) and dropping items. Includes a dialogue system (dialogues) for NPC communication, with direction-based reactions.
Pathfinding and Movement:

Implements entity movement based on directions and random actions using the setAction method. Handles pathfinding for navigating around obstacles and other entities.

Combat and Damage:

Manages combat by calculating damage (damagePlayer), handling invincibility periods, and shooting projectiles. Includes particle effects for visual damage feedback (generateParticle).

Graphics and Rendering:

Renders entities on screen with proper positioning and animations. Handles visual effects such as alpha transparency for invincibility (changeAlpha) and dying animations (dyingAnimation). Also includes a health bar for monsters.

Utility Methods:

Includes utility methods for tasks like scaling images (setup) and managing particle attributes (getParticleColor, getParticleSize, etc.).

3. Player.java:

Movement and Direction Handling:

The player's position and direction are controlled via the KeyHandler class (upPressed, downPressed, etc.). Collision with tiles, objects, NPCs, monsters, and interactive tiles is handled with methods like checkTile, checkObject, checkEntity, etc.

Attacking and Combat System:

Attacks are based on the weapon equipped, and the attacking() method handles the animation and damage application. Monsters can be damaged, and specific interactions with destructible tiles (like trees) are included.

Inventory Management:

The inventory stores items the player collects, with a maximum size of 20. Items can be weapons, consumables, or interactable objects.

Leveling Up:

Gaining experience allows the player to level up, increasing stats like maxLife, strength, and dexterity. When leveling up, appropriate sound effects and dialogue are triggered.

Graphics and Animation:

The draw() method renders the player sprite based on direction, action (moving/attacking), and animation state (spriteNum). Alpha transparency (invincible) is used to show temporary invincibility.

Resource Handling:

Ammunition, mana, and health are managed, with limits on their maximum values. A projectile object handles ranged attacks like snowballs.

Interaction with NPCs:

Allows the player to interact with NPCs, triggering dialogue sequences. Transitions to the dialogue state within the game.

Modularize Methods for Readability:

Some methods like update() and attacking() are lengthy and could be split into smaller, more focused methods. For example: Create separate methods for collision handling, attack initiation, and key input processing.

4. GamePanel.java:

Screen and Window Settings:

Defines parameters for the game's display, including tile sizes (originalTileSize, scale, tileSize), screen dimensions (screenWidth, screenHeight), and window modes such as full-screen via setFullScreen().

World Map Configuration:

Manages the game's world map by specifying the maximum world dimensions (maxWorldCol, maxWorldRow).

Includes logic to represent elements in the environment as tiles and handles their graphical representation through TileManager.

Entity and Object Management:

Efficiently manages players, NPCs, monsters, interactive tiles, and other entities using arrays and lists (npc[], obj[], snowman[], projectileList, etc.).

Organizes and updates entities based on types and states (e.g., alive, dying).

Uses sorting mechanisms (Collections.sort) for rendering entities based on their position (worldY) to ensure proper layer visibility.

Game State Handling:

Implements a state management system for handling transitions between different game states:

- titleState: Title screen.
- playState: Main gameplay.
- pauseState: Pause mode.
- dialogueState, characterState, optionsState, gameOverState: Various other modes.

Game Loop

The run() method implements a consistent game loop:

- Controls the game's frame rate (FPS) for smooth gameplay.
- Updates the game's logic and renders frames via update() and repaint().

Graphics Rendering and Debugging

Handles rendering of tiles, entities, interactive elements, and the user interface (paintComponent()).

Incorporates debugging features to display real-time statistics such as player position and frame rendering time when debugging is enabled.

Audio Integration

Seamlessly incorporates sound effects and background music using Sound class objects (music and se).

Provides methods:

- playMusic(int i): Plays background music.
- stopMusic(): Stops the current music.
- playSE(int i): Plays sound effects.

Additional Functionalities

Game Setup and Reset:

- setupGame(): Initializes objects, NPCs, and interactive tiles.
- retry() and restart(): Reset game states, player attributes, and reinitialize entities.

5. CollisionChecker.java:

The class is designed to handle collision detection in the game, ensuring that entities do not pass through objects, walls, or other entities in the game world. This is a critical component for maintaining realism and interaction within the game.

6. AssetSetter.java:

Object Initialization:

Creates instances of various game objects, such as: OBJ_Axe, OBJ_Potion_Red, OBJ_Heart, OBJ_ManaCrystal, OBJ_Boot, and OBJ_Chest.

Assigns specific world coordinates to these objects for proper placement in the game environment using the gp.obj array.

Object Placement on the Map:

Objects are added to the gp.obj array with their tile-based coordinates calculated using gp.tileSize to ensure proper alignment within the game world.

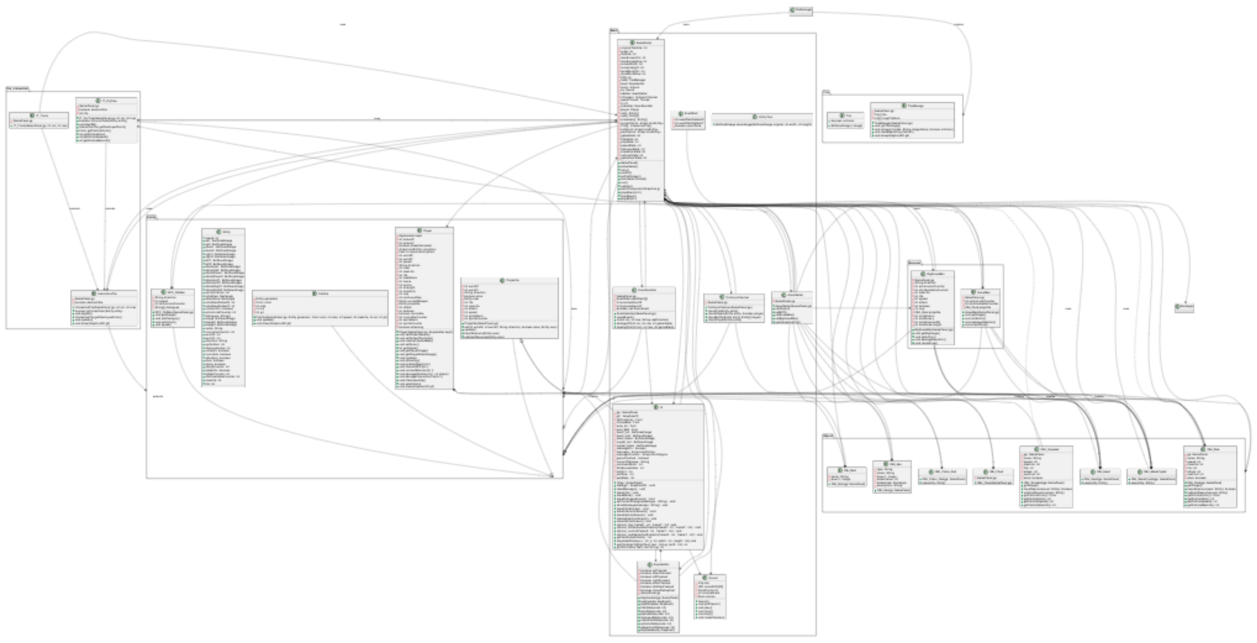
NPC and Monster Initialization:

Creates and positions instances of NPCs and monsters within the game world, assigning precise tile-based coordinates to define their starting locations on the map.

Interactive Tile Initialization:
Generates and places interactive tiles on the map, assigning specific coordinates to establish their exact locations within the game world.

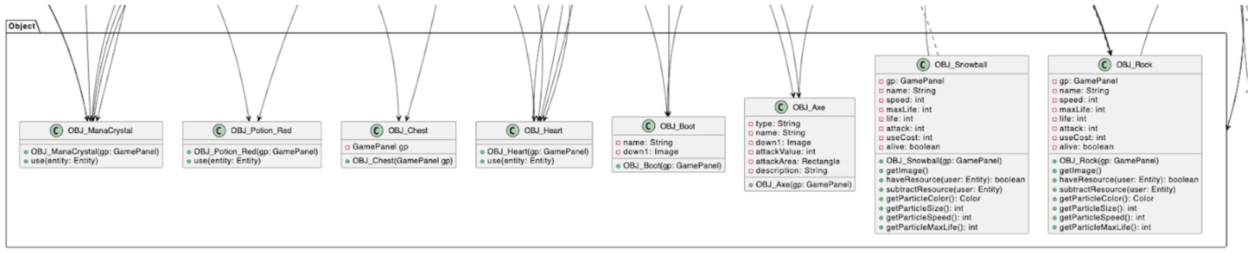
CHAPTER 4: UML CLASS DIAGRAM

There are 6 main packages in our group's UML diagram, which are presented respectively below:

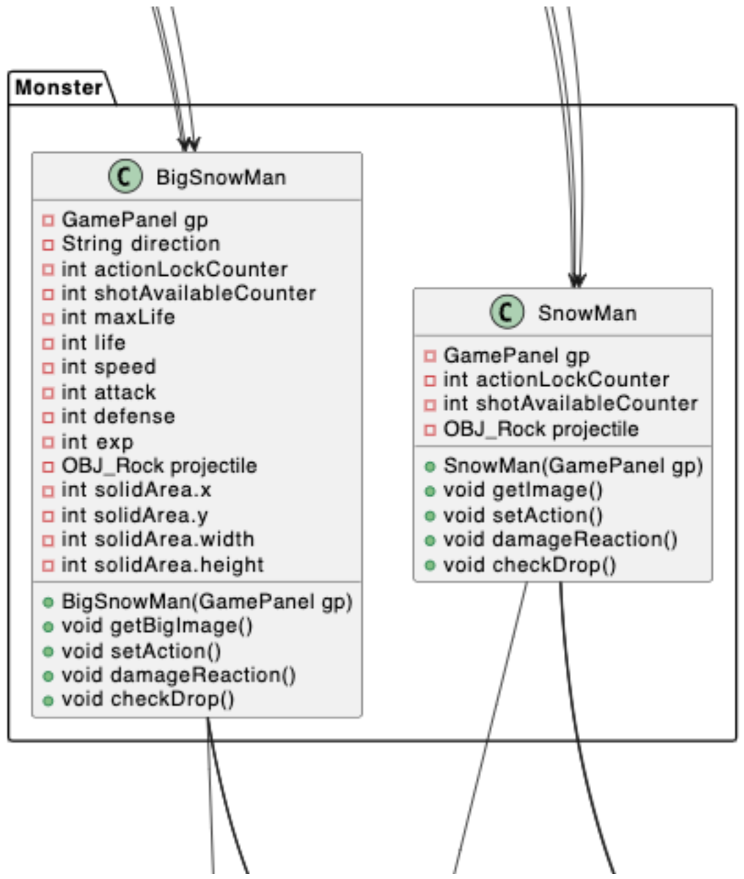


1.Object

This package is an encapsulation of a variety of game objects with specific functionalities. The use of a common superclass *(SuperObject)* allows for a consistent interface and potentially shared behaviors among these game objects.

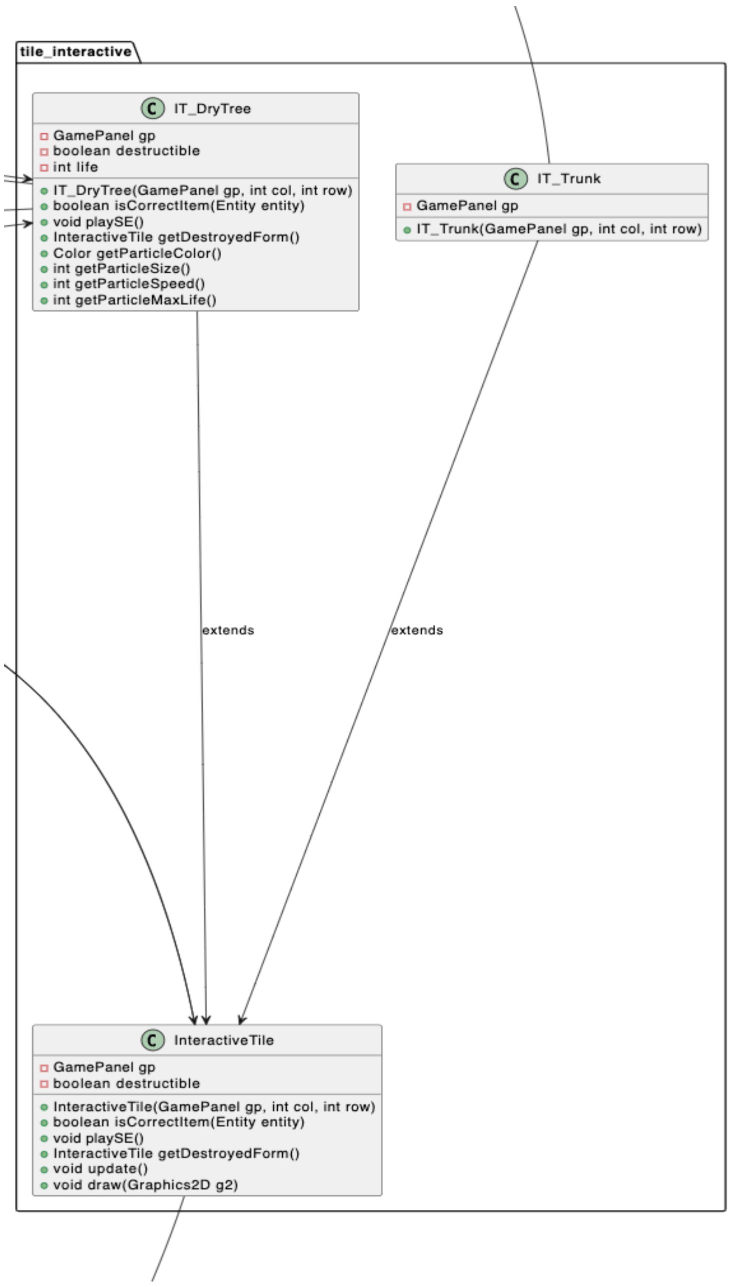


2.Monster



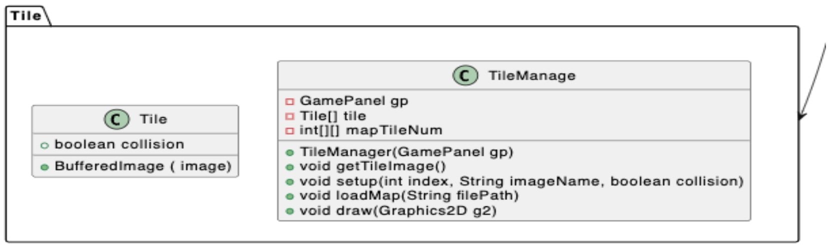
The Snowman class represents a snowman entity within the monster package. It encapsulates attributes related to the snowman's behavior, movement, and interactions. The methods are responsible for updating its state, managing actions, and handling reactions to various game events. The class operates within the broader context of the game's monster-related functionalities. The BigSnowMan class has the same attributes with Snowman.However, it has stronger speed, attack.

3. **Interactive Tiles (tile_interactive)**



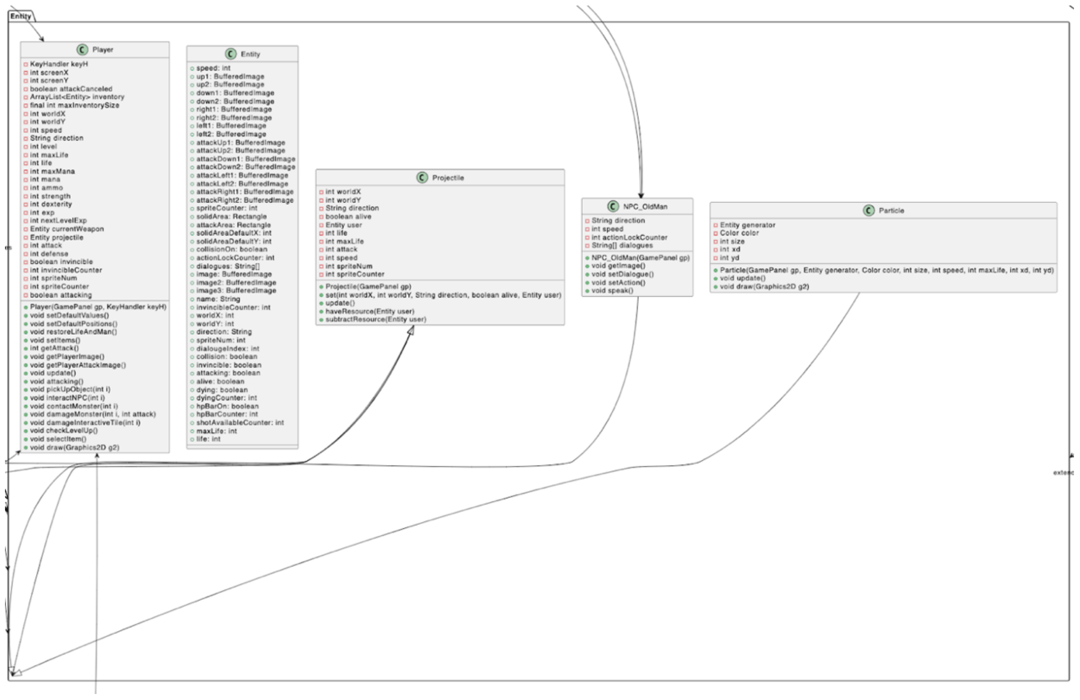
This package defines a set of classes related to interactive tiles in a game. The *InteractiveTile* class serves as a base with common attributes and methods, while specialized tiles *(IT_DryTree, IT_Trunk)* inherit from it, providing specific implementations and additional methods. The use of encapsulation and inheritance promotes modularity and extensibility in handling different types of interactive tiles.

4.Tile

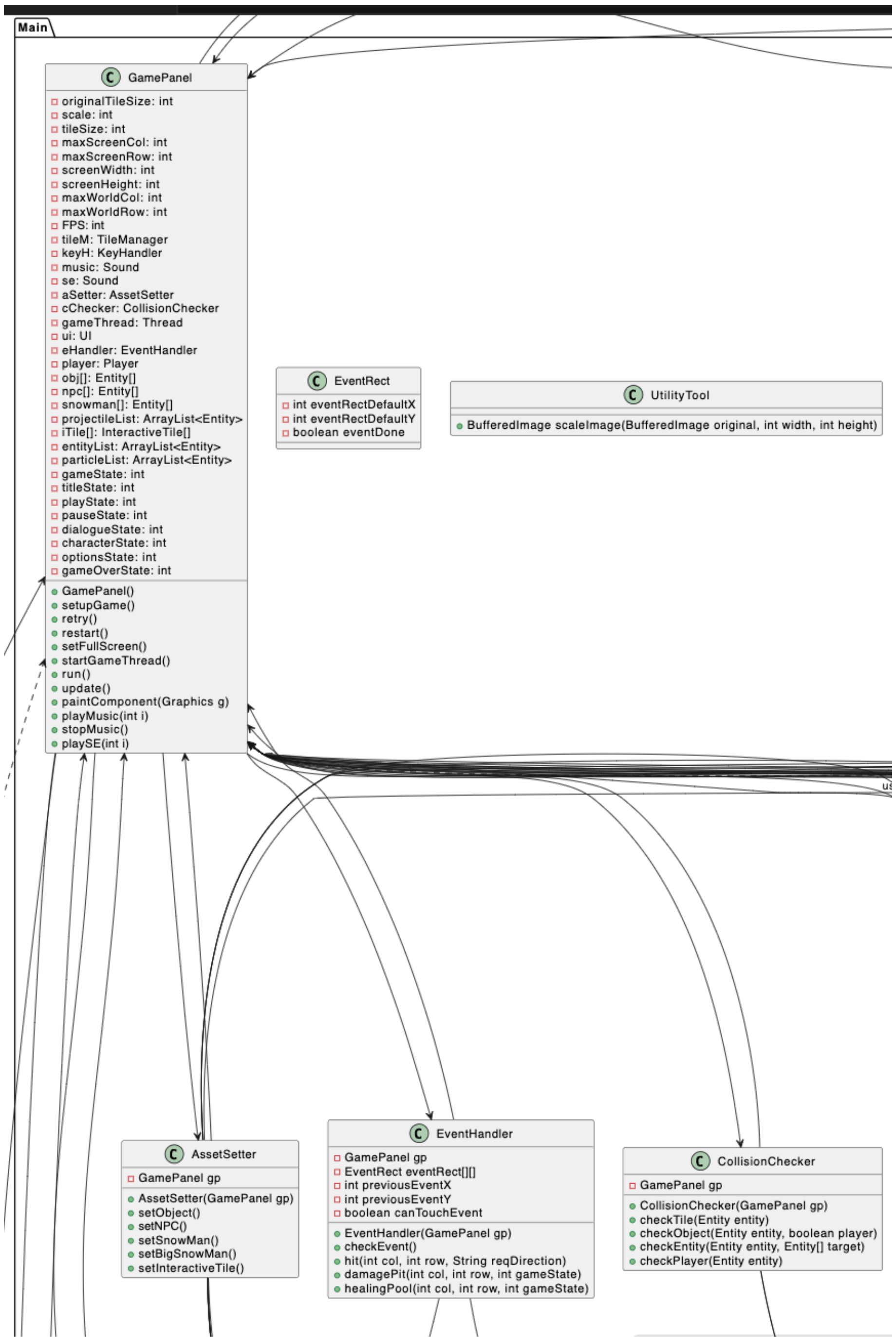


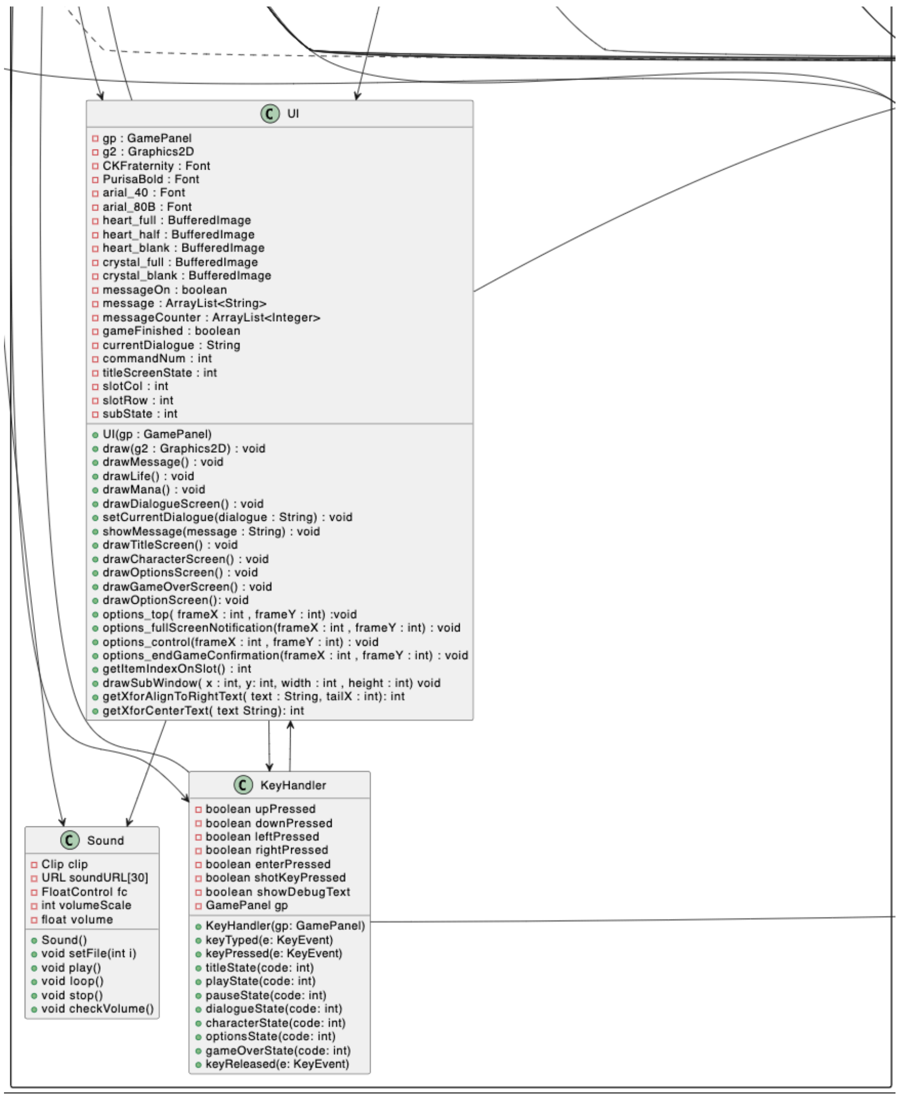
This package is designed for managing and rendering tiles in a game environment. The *Tile* class encapsulates individual tile properties, while the *TileManager* class handles the overall management, setup, and drawing of tiles on the game panel.
5.Entity

This package offers a flexible framework to manage game entities, allowing easy customization of behaviors, interactions, and visual effects.

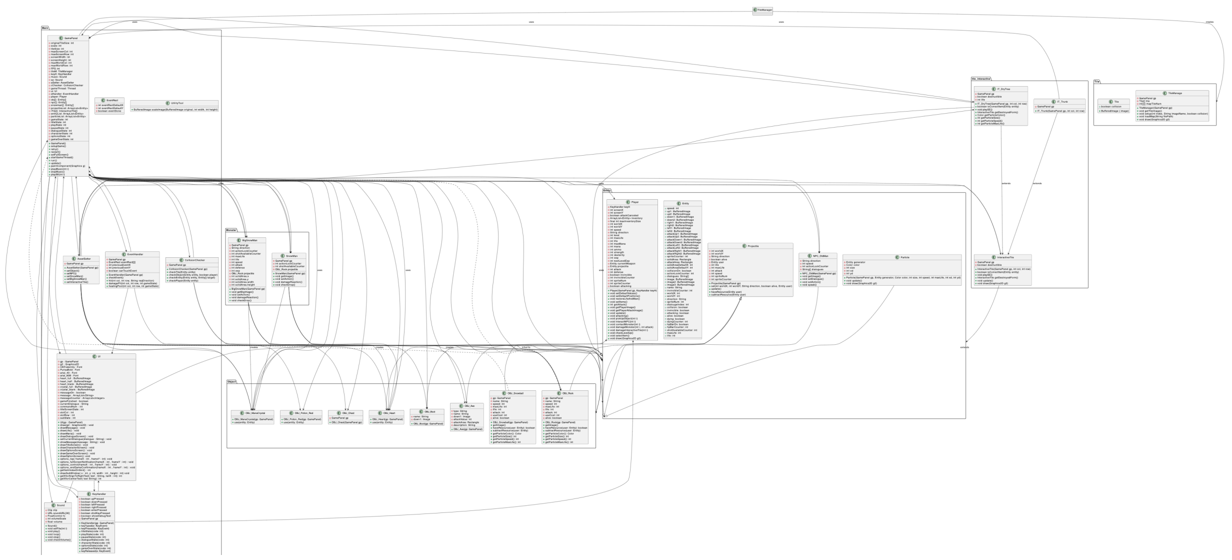


6.Main


Main
GamePanel
originalTileSize: int
scale: int
tileSize: int
maxScreenCol: int
maxScreenRow: int
screenWidth: int
screenHeight: int
maxWorldCol: int
maxWorldRow: int
FPS: int
tileM: TileManager
keyH: KeyHandler
music: Sound
se: Sound
aSetter: AssetSetter
cChecker: CollisionChecker
gameThread: Thread
ui: UI
eHandler: EventHandler
player: Player
obj[]: Entity[]
npc[]: Entity[]
snowman[]: Entity[]
projectileList: ArrayList<Entity>
iTile[]: InteractiveTile[]
entityList: ArrayList<Entity>
particleList: ArrayList<Entity>
gameState: int
titleState: int
playState: int
pauseState: int
dialogueState: int
characterState: int
optionsState: int
gameOverState: int
GamePanel()
setupGame()
retry()
restart()
setFullScreen()
startGameThread()
run()
update()
paintComponent(Graphics g)
playMusic(int i)
stopMusic()
playSE(int i)
EventRect
int eventRectDefaultX
int eventRectDefaultY
boolean eventDone
UtilityTool
BufferedImage scaleImage(BufferedImage original, int width, int height)
AssetSetter
GamePanel gp
AssetSetter(GamePanel gp)
setObject()
setNPC()
setSnowMan()
setBigSnowMan()
setInteractiveTile()
EventHandler
GamePanel gp
EventRect eventRect[][]
int previousEventX
int previousEventY
boolean canTouchEvent
EventHandler(GamePanel gp)
checkEvent()
hit(int col, int row, String reqDirection)
damagePit(int col, int row, int gameState)
healingPool(int col, int row, int gameState)
CollisionChecker
GamePanel gp
CollisionChecker(GamePanel gp)
checkTile(Entity entity)
checkObject(Entity entity, boolean player)
checkEntity(Entity entity, Entity[] target)
checkPlayer(Entity entity)


UI
gp : GamePanel
g2 : Graphics2D
CKFraternity : Font
PurisaBold : Font
arial_40 : Font
arial_80B : Font
heart_full : BufferedImage
heart_half : BufferedImage
heart_blank : BufferedImage
crystal_full : BufferedImage
crystal_blank : BufferedImage
messageOn : boolean
message : ArrayList<String>
messageCounter : ArrayList<Integer>
gameFinished : boolean
currentDialogue : String
commandNum : int
titleScreenState : int
slotCol : int
slotRow : int
subState : int
UI(gp : GamePanel)
draw(g2 : Graphics2D) : void
drawMessage() : void
drawLife() : void
drawMana() : void
drawDialogueScreen() : void
setCurrentDialogue(dialogue : String) : void
showMessage(message : String) : void
drawTitleScreen() : void
drawCharacterScreen() : void
drawOptionsScreen() : void
drawGameOverScreen() : void
drawOptionScreen(): void
options_top(frameX : int , frameY : int) :void
options_fullScreenNotification(frameX : int , frameY : int) : void
options_control(frameX : int , frameY : int) : void
options_endGameConfirmation(frameX : int , frameY : int) : void
getItemIndexOnSlot() : int
drawSubWindow(x : int, y: int, width : int , height : int) void
getXforAlignToRightText(text : String, tailX : int): int
getXforCenterText(text String): int
KeyHandler
boolean upPressed
boolean downPressed
boolean leftPressed
boolean rightPressed
boolean enterPressed
boolean shotKeyPressed
boolean showDebugText
GamePanel gp
KeyHandler(gp: GamePanel)
keyTyped(e: KeyEvent)
keyPressed(e: KeyEvent)
titleState(code: int)
playState(code: int)
pauseState(code: int)
dialogueState(code: int)
characterState(code: int)
optionsState(code: int)
gameOverState(code: int)
keyReleased(e: KeyEvent)
Sound
Clip clip
URL soundURL[30]
FloatControl fc
int volumeScale
float volume
Sound()
void setFile(int i)
void play()
void loop()
void stop()
void checkVolume()

The *main* package contains classes responsible for setting up and managing various aspects of the game, including assets, collision checking, configuration, event handling, UI, sound, and key interactions. The package design allows for modular and organized management of different game functionalities.



CHAPTER 5: EVALUATION

1. Performance Evaluation:

Quick Response Time:

Because we maintained the proper link between images during each motion of the characters, the game responds quickly, providing players with rapid feedback and a seamless interactive experience during smooth gameplay.

Memory Usage:

We optimized our game to use resources efficiently, and the average memory use is 24 MB.

Load Game:

Our game's initial load time is about 2 seconds, from activating the application to starting gaming. This short load time helps gamers to get right into the action without any additional pauses.

2. **Code Quality**

Readability: The codes rigidly adhere to well-organized paths from the start, ensuring that they function correctly as predicted. So the arrangements in these codes are easy to understand.

Maintainability and code comments: To make it easier to identify objects and classes, we have provided uniform naming schemes. We have also incorporated comments into our code to clarify the function and goal of complex code. Other developers will be able to rapidly understand our code thanks to this.

CHAPTER 6: EXPERIENCE

Developing "*Christmas Treasure*" has been a deeply enriching and rewarding experience for our team. This project allowed us to merge creativity with technical skills, pushing the boundaries of our understanding of Object-Oriented Programming. Throughout this journey, we gained invaluable insights and skills that will undoubtedly aid us in future endeavors.

Moreover, *"Christmas Treasure"* was more than just a project—it was a journey of growth, creativity, and learning. The skills and experiences we gained will serve as a strong foundation as we move forward, not only in game development but also in tackling real-world programming challenges. We are proud of what we achieved and look forward to building upon this experience in future projects.

Finally, we would like to express our deepest gratitude to our tutor, Mr. Nghia, for guiding us through both the Object - Oriented Programiboratories.