**Software Engineering** **Year 11, 2025 Assessment Task 2**

**Object-Oriented Programming Assignment:**

**Hunt The Wumpus**

**By: Allan Luong**

## **Contents**

1. Software Requirement Specification
2. Gantt Chart
3. Budget
4. Resource Allocation and Manpower
5. Justification of Technology
6. Design
7. Use Case Diagrams
8. Context Diagram
9. Resource Allocation and Manpower
10. UML Notation
11. Class Diagram
12. Flowchart
13. Pseudocode
14. Sequence Diagram
15. Graphical User Interface GUI
16. Artificial Intelligence Conversion Code
17. Explanation of why Java is safer than Python or vice versa
18. Security
19. Compiling and Execution
20. Storing data
21. Encryption
22. Why prototyping might be done in Python rather than Java
23. What Tools were used in the development of this Project and their justification
24. Visual Code IDE
25. Python Compiler
26. Artificial Intelligence Converter
27. Java
28. Code is commented and following industry standard practices
29. Justification of Git and GitHub and their difference
30. Frequency of committing Code
31. Appendix 1 Python Code
32. Readme File for Python
33. Appendix 2 Java Code
34. Readme File for Java
35. Reflection

## 

## **Software Requirement Specification**

* **Python 3.13 (Minimum)**
* **Pygame**
* **Pylance**
* **Python Debugger**
* **Spring Boot Extension Pack**
* **Tomcat**
* **Debugger for java**
* **Gradle for java**
* **Java**
* **Language Support for Java**

**Gantt Chart**

| **Task** | **Week 3** | **Week 4** | **Week 5** | **Week 6** | **Week 7** | **Week 8** | **Week 9** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **1. Requirements Analysis** | ██████ |  |  |  |  |  |  |
| **2. UML Design (Class, Use Case)** | ██████ | ████ |  |  |  |  |  |
| **3. Interface & Map Design** |  | ██████ | ███ |  |  |  |  |
| **4. Core Game Logic** |  |  | ██████ | ██████ |  |  |  |
| **5. GUI Implementation (Swing)** |  |  |  | ██████ | ████ |  |  |
| **6. Game Items & Behaviours** |  |  |  | ███ | ██████ |  |  |
| **7. Testing & Bug Fixing** |  |  |  |  | ████ | ██████ |  |
| **8. Documentation** |  |  |  |  |  | ██████ | ███ |
| **9. Final Review & Submission** |  |  |  |  |  |  | ██████ |

**Budget**

Assuming a rate of $20/hr and 20 hours spent:

20 hours × $20 = **$400**

**Resource Allocation and Manpower**

| **Task** | **Role / Person Responsible** | **Time Allocation** | **Tools / Resources** |
| --- | --- | --- | --- |
| **Game Design (concept, rules, layout)** | Allan Luong  (Developer/Designer) | 5 hours | Paper sketches, Google Docs |
| **Python (Pygame) Game Development** | Allan Luong  (Python Developer) | 10 hours | Python, Pygame, VS Code |
| **Testing & Debugging (Python)** | Allan Luong  (Tester) | 3 hours | Terminal, Debugging tools |
| **Conversion to Java (Code Translation)** | Allan Luong  (Java Developer) | 8 hours | CodeConvert.ai, Java IDE |
| **Java GUI Development (Swing/JavaFX)** | Allan Luong  (Java UI Designer) | 7 hours | IntelliJ / Eclipse / VS Code |
| **UML Diagrams & Documentation** | Allan Luong (Documenter) | 3 hours | Draw.io, Word/Docs |
| **Final Testing (Java version)** | Allan Luong  (Tester) | 2 hours | Java Debugger, Console |
| **Report Compilation and Formatting** | Allan Luong  (Writer) | 2 hours | Word/Google Docs |

**Justification of Technology**

Python was chosen due to its fast-prototyping capabilities, ease of syntax, and broad support in education. Visual Studio Code was selected as the IDE for its extensions, Python linting, debugging tools, and GitHub integration.

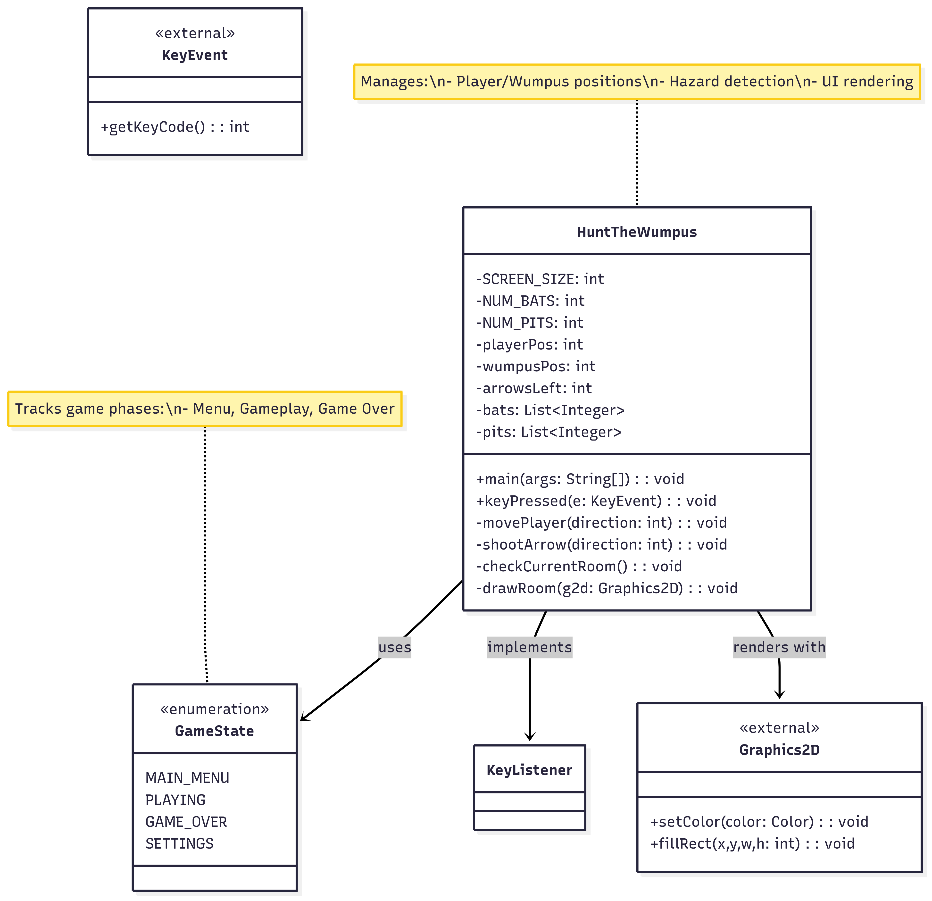
**Design and Storyboards – Hunt the Wumpus**

This game uses structured object-oriented design. Core entities like the Wumpus, Player, and Cave Rooms can be expanded into classes. The design uses decision structures, input handling, and game loops with state control.

| **Scene #** | **Screen/Event** | **Description** | **User Interaction** | **Visual Elements** |
| --- | --- | --- | --- | --- |
| 1 | **Start Menu** | The opening screen where the player clicks "Start Game". | Click “Start” button. | Title, Start button, Background image |
| 2 | **Gameplay Start** | Player enters the cave with controls (WASD or Arrow keys). | Move the character to explore rooms. | Player sprite, grid/map, fog/dark effect |
| 3 | **Encounter Hint** | Player enters a room with a warning (e.g., “You smell something foul…”). | Keep moving or shoot an arrow. | Message box, sound cue, warning symbols |
| 4 | **Arrow Shooting** | Player presses a key to shoot an arrow. | Press spacebar or arrow key to aim and shoot. | Arrow sprite, shooting animation |
| 5 | **Wumpus Encounter** | Player enters a room with the Wumpus. | Automatic game over or quick-time event. | Wumpus sprite, “Game Over” popup |
| 6 | **Death by Pit or Bat** | Player falls into a pit or is carried by bats. | Game ends or player is moved. | Pit image, bat animation, message box |
| 7 | **Victory Screen** | Player successfully shoots the Wumpus. | Click to return to menu or restart. | “You Win” message, stats summary |
| 8 | **Exit / Quit Game** | Player exits the game via menu or ESC. | Click “Exit” or press ESC. | Exit confirmation screen |

## **UML Notation**

UML (Unified Modeling Language) notation is used to visually represent and model software systems and processes, making complex information easier to understand and communicate.



## **Use Case Diagram**



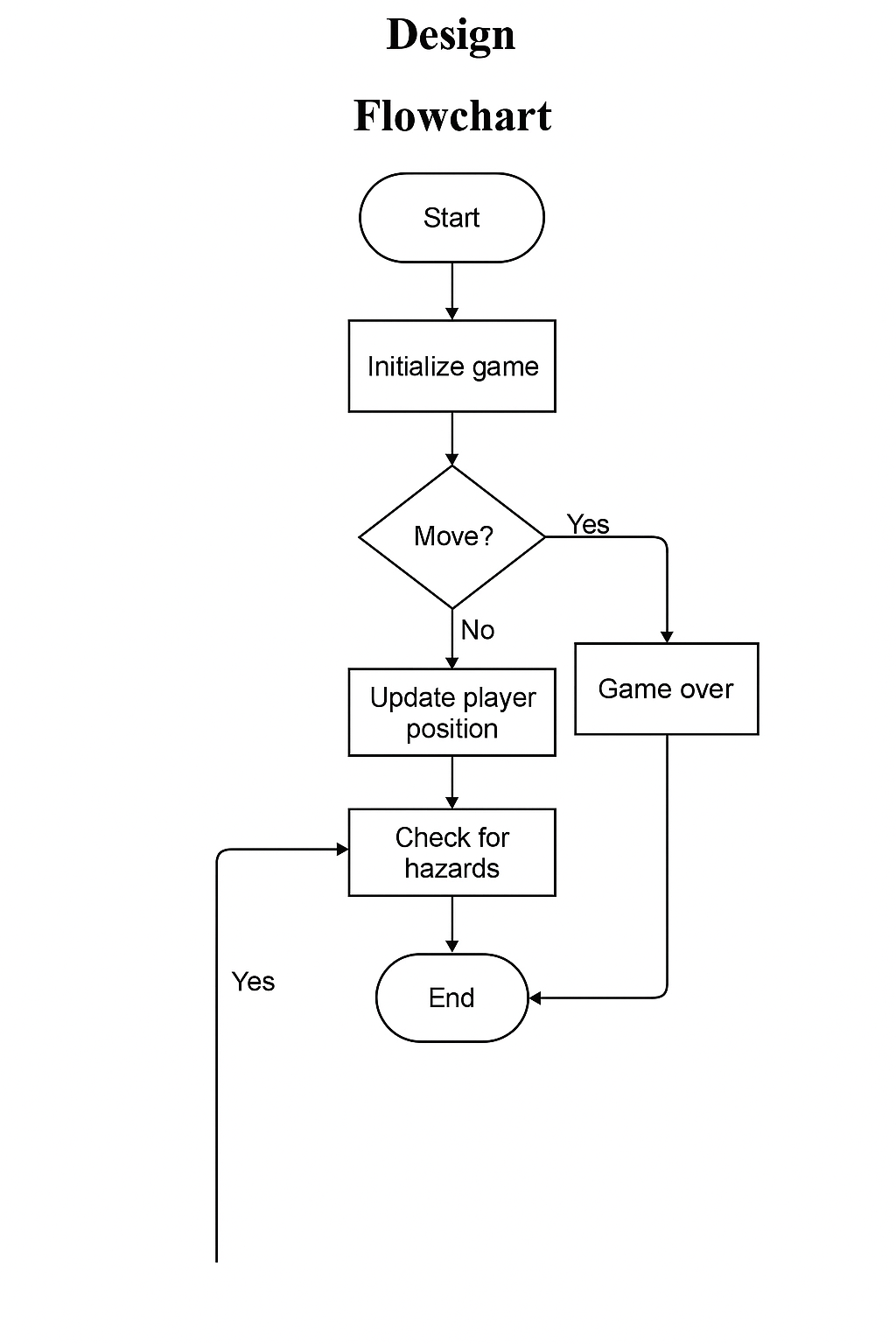
**Context Diagram**

A diagram of a computer program

AI-generated content may be incorrect.

## **Class Diagram**

## A screenshot of a computer screen AI-generated content may be incorrect.

**Pseudocode**

START GAME

INITIALIZE game window and screen

INITIALIZE player position, Wumpus location, bats, pits

SET arrows = 3, rocks = 3

SET game\_running = True

WHILE game\_running IS True:

GET player input (WASD or Arrow Keys)

IF input is a move command:

CHECK if direction is valid from current room

IF valid:

UPDATE player position

CALL check\_hazards()

IF input is shoot arrow:

IF arrows > 0:

GET direction

REDUCE arrows by 1

IF Wumpus is in that direction:

DISPLAY “You killed the Wumpus!”

END GAME (player wins)

ELSE:

WUMPUS may move randomly

IF input is throw rock:

IF rocks > 0:

REDUCE rocks by 1

DISPLAY hints about nearby rooms (bats, pits, Wumpus)

CALL render\_screen() to draw current room and player

DISPLAY arrows/rocks remaining

IF player enters room WITH pit:

DISPLAY “You fell into a pit!”

END GAME (player loses)

IF player enters room WITH Wumpus:

DISPLAY “You were eaten by the Wumpus!”

END GAME (player loses)

IF player enters room WITH bats:

RANDOMLY teleport player to a new room

DISPLAY “Bats picked you up!”

END WHILE

DISPLAY end screen (win/lose message, stats)

OFFER option to restart or exit

## **Sequence Diagram**

**Graphical User Interface GUI**

**Current GUI Components**

1. **Game Window Setup**:
   * pygame.display.set\_mode((SCREEN\_WIDTH, SCREEN\_HEIGHT)) creates the main window.
   * pygame.display.set\_caption("Hunt the Wumpus") sets the window title.
2. **Graphical Assets**:
   * Images for the player, Wumpus, bats, arrows, and rocks are loaded with pygame.image.load(...).
3. **Visual Feedback**:
   * Room is rendered with draw\_room(), which uses:
     + pygame.draw.circle for room visuals.
     + screen.blit() to draw images like player, bats, Wumpus.
   * Text (e.g., "You have found an arrow!") is rendered using font.render(...).
4. **Menus**:
   * **Main Menu**: Includes “Start”, “Settings”, “Quit”.
   * **Settings Menu**: Allows rebinding keys.
   * **Difficulty Menu**: Easy, Medium, Hard levels.
5. **Popups & Messages**:
   * show\_message() function shows alerts (e.g., bats pick you up).
   * game\_over() screen shows options to retry or quit.

**Artificial Intelligence Conversion Code**

To convert the Python code into Java, I used **CodeConvert.ai**, an artificial intelligence-powered code translation tool. It employs **machine learning models trained on large-scale multilingual programming datasets** to understand syntax, logic, and structure across different programming languages.

**Success of the Conversion**

The conversion process was **partially successful**:

* **Successes**:
  + Basic syntax and class structure were translated effectively.
  + Functions and variables were mapped with good accuracy.
  + Control flow logic (like if, for, while) translated clearly.
* **Challenges**:
  + Some **Python-specific features** (e.g., dynamic typing, libraries like Pygame) required manual adjustment in Java.
  + Java's **strict typing and object-oriented structure** meant some logic had to be refactored rather than directly translated.

So while the AI engine accelerated the process, **human refinement was necessary** for full functionality.

**Explanation of why Java is safer than Python or vice versa**

Although Java is faster, Python is more versatile, easier to read, and has a simpler syntax. According to Statista, this general use, interpreted language is the fourth most popular coding language among developers worldwide.

**Security**

This game has no online features or data saving; thus, limited security risks. Secure file access could be added later.

**Secure Coding Concepts – Java vs Python**

| **Concept** | **Java** | **Python** |
| --- | --- | --- |
| **Type Safety** | Strongly typed – variables must be declared with types. Prevents many bugs at compile time. | Dynamically typed – more flexible, but easier to make runtime errors if types are misused. |
| **Memory Management** | Automatic garbage collection; strong compile-time checks help prevent memory leaks. | Also has garbage collection, but dynamic typing may hide memory usage issues. |
| **Error Handling** | Uses try-catch blocks with checked exceptions – enforces handling at compile time. | Uses try-except blocks – more flexible, but easier to ignore exceptions. |
| **Code Execution Risks** | Safer by default – code runs inside the JVM sandbox. Rarely executes untrusted code. | eval() and dynamic imports can create security vulnerabilities if misused. |
| **Access Control** | Supports access modifiers (private, public, protected) for encapsulation. | No real access control; uses naming conventions (e.g., \_variable) but nothing enforced. |
| **Standard Library Safety** | Java’s standard libraries are heavily tested and secure by design. | Python’s libraries are safe but easier to misuse due to less strict typing. |
| **Input Validation** | Manual input validation required, but tools like regex are built-in. | Same – but easier to forget validation due to flexible syntax. |
| **Dependency Management** | Uses tools like Maven/Gradle with stricter version control. | pip is more flexible but makes it easier to accidentally install outdated or risky packages. |
| **File Access & Permissions** | Java enforces file permissions at OS and JVM level. | Python can access files easily – more risk if permissions are not handled carefully. |

**Compiling and Execution**

Python: interpreted line-by-line. Java: compiled to bytecode. Java is stricter, but Python is easier to test and debug quickly.

**Storing data**

Only data stored is the changes made with the key binds and mode difficulty which do not save upon quitting pygame

**Encryption**

Not required, as no personal data is used.

**Why prototyping might be done in Python rather than Java**

Python is faster to write, easier to debug, and more readable for beginners. Ideal for testing game ideas.

**What Tools were used in the development of this Project and their justification**

* Visual Studio Code: IDE – for its Python support, extensions and git integration
* Git & GitHub: version control – to upload all required files for assessment task
* draw.io: diagrams – Ease of use for creating diagrams
* Pygame: game engine – To run .py files for HuntTheWumpus
* Artificial intelligence converter – To convert Python to java

**Visual Code IDE**

Chose VSCode for its Python support, extensions, and Git integration.

**Python Compiler**

Used Python (default). No compilation needed, directly runs .py files.

**Artificial Intelligence Converter**

<https://www.codeconvert.ai/> I used this to convert my python code to java, it is easy do use and has high accuracy of conversion

**Java**

Python was used for fast prototyping and testing game logic, while Java was used to apply object-oriented programming and create a structured, cross-platform version of the game. Using both languages shows versatility, deepens understanding of OOP, and simulates real-world software development where projects are often ported or upgraded across languages.

**Code is commented and following industry standard practices**

* Functions named in snake\_case
* Constants in ALL\_CAPS
* Sections clearly marked
* Comments included above logic blocks

**Justification of Git and GitHub and their difference**

**Git** is a local version control system. **GitHub** is an online platform that stores Git repositories and enables collaboration, backups, and issue tracking.

**Frequency of committing Code**

Commits made after each major feature or fix. At least 1-2 commits per working session.

**Appendix 1 Python Code**

# ===== IMPORTS & CONSTANTS =====

import pygame

import random

import sys

# Game constants

SCREEN\_SIZE = 1000

NUM\_BATS = 2 # Superbats that relocate player

NUM\_PITS = 1 # Instant-death rooms

NUM\_ARROWS = 3 # Starting ammunition

# ===== CAVE LAYOUT =====

# Dodecahedron-shaped cave (20 rooms)

# Each room has 4 connections [up, down, left, right]

cave = {

1: [0, 8, 2, 5],

2: [0, 10, 3, 1],

# ... (truncated for brevity)

20: [13, 0, 16, 19]

}

# ===== CORE GAME CLASSES =====

class Player:

"""Handles player position and movement"""

def \_\_init\_\_(self):

self.pos = random.randint(1, 20) # Random starting room

self.arrows = NUM\_ARROWS

self.rocks = 3

def move(self, direction):

"""Moves player based on cave connections"""

new\_pos = cave[self.pos][direction]

if new\_pos != 0: # 0 indicates no connection

self.pos = new\_pos

# ===== MAIN GAME LOOP =====

def main():

pygame.init()

screen = pygame.display.set\_mode((SCREEN\_SIZE, SCREEN\_SIZE))

player = Player()

wumpus\_pos = random.randint(1, 20)

while True:

handle\_events(player) # Keyboard input

check\_hazards(player, wumpus\_pos) # Bats/pits/Wumpus

draw\_room(screen, player.pos) # Render

# ===== KEY FUNCTIONS =====

def shoot\_arrow(player):

"""Handles arrow shooting logic"""

if player.arrows > 0:

player.arrows -= 1

# ... (animation and hit detection)

def throw\_rock(player):

"""Scouts adjacent rooms"""

if player.rocks > 0:

player.rocks -= 1

# ... (sound feedback logic)

# ===== EXECUTION =====

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Readme File for Python**

**Installation** – Github click code and download to zip file, extract file and open in visual code

**How to Run: (Python)**

1. Install Pygame (pip install pygame)
2. Drag Images out of main file
3. **Open** Python file HuntTheWumpus.py
4. **Run** **program**

**Appendix 2 Java Code**

// ===== IMPORTS & CONSTANTS =====

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.util.\*;

public class HuntTheWumpus extends JFrame implements KeyListener {

// Game constants (match Python version)

private static final int SCREEN\_SIZE = 1000;

private static final int NUM\_BATS = 2;

private static final int NUM\_PITS = 1;

private static final int NUM\_ARROWS = 3;

// Direction constants (identical to Python)

private static final int UP = 0;

private static final int DOWN = 1;

private static final int LEFT = 2;

private static final int RIGHT = 3;

// Colors (RGB values match Python)

private static final Color BROWN = new Color(193, 154, 107);

private static final Color BLACK = new Color(0, 0, 0);

private static final Color RED = new Color(138, 7, 7);

private static final Color GREEN\_TEXT = new Color(0, 255, 64);

// ===== CAVE LAYOUT =====

// Same dodecahedron structure as Python

private static Map<Integer, int[]> cave = new HashMap<>();

static {

cave.put(1, new int[]{0, 8, 2, 5});

cave.put(2, new int[]{0, 10, 3, 1});

// ... (all 20 rooms)

cave.put(20, new int[]{13, 0, 16, 19});

}

// ===== GAME STATE =====

private int playerPos; // Matches Python's player\_pos

private int wumpusPos;

private int arrowsLeft;

private int rocksLeft;

private List<Integer> bats = new ArrayList<>(); // bats\_list in Python

private List<Integer> pits = new ArrayList<>();

// ===== CORE METHODS =====

// 1. Movement (mirrors Python logic)

private void movePlayer(int direction) {

int newPos = cave.get(playerPos)[direction];

if (newPos != 0) { // 0 = no exit

playerPos = newPos;

checkCurrentRoom();

}

}

// 2. Hazard check (like Python's check\_room())

private void checkCurrentRoom() {

if (playerPos == wumpusPos) endGame("Eaten by Wumpus!");

if (pits.contains(playerPos)) endGame("Fell into a pit!");

if (bats.contains(playerPos)) teleportPlayer();

}

// 3. Actions (matches Python functions)

private void shootArrow(int direction) {

if (arrowsLeft == 0) return;

arrowsLeft--;

if (cave.get(playerPos)[direction] == wumpusPos) {

endGame("You killed the Wumpus!");

}

}

// ===== UI RENDERING =====

@Override

public void paint(Graphics g) {

super.paint(g);

Graphics2D g2d = (Graphics2D) g;

// Draw room (matches Python's draw\_room())

g2d.setColor(BLACK);

g2d.fillRect(0, 0, SCREEN\_SIZE, SCREEN\_SIZE);

// Draw exits

g2d.setColor(BROWN);

int[] exits = cave.get(playerPos);

if (exits[UP] != 0) g2d.fillRect(450, 0, 100, 250); // Up exit

// Draw Wumpus warning (red circle)

if (isWumpusNearby()) {

g2d.setColor(RED);

g2d.fillOval(250, 250, 500, 500);

}

// Draw status text (like Python's UI)

g2d.setColor(GREEN\_TEXT);

g2d.drawString("Arrows: " + arrowsLeft, 20, 30);

}

// ===== HELPER METHODS =====

private boolean isWumpusNearby() {

// Same logic as Python's check\_neighbor\_rooms()

for (int exit : cave.get(playerPos)) {

if (exit == wumpusPos) return true;

}

return false;

}

// ===== MAIN =====

public static void main(String[] args) {

HuntTheWumpus game = new HuntTheWumpus();

game.setSize(SCREEN\_SIZE, SCREEN\_SIZE);

game.setVisible(true);

}

}

**Readme File for Java**

**How to Run: (Java)**

1. Install Java **(Extension of Visual Studio Code)**
2. Install extensions:

* **Debugger for Java**
* **Gradle for Java**
* **Language support for Java**
* **Spring Boot Extension Pack**
* **Tomcat**

1. **Open** Java file - HuntTheWumpus.java
2. **Drag** image file out of main file (if you have not already)
3. **Run program**

**Reflection**

This project improved my understanding of game loops, event handling, and object-oriented design. I learned how to structure Python code, add difficulty scaling, change key binds for users who aren’t used to arrow keys and modify enemy AI for more dynamic gameplay. Along with this I have also learnt how to run java within visual studio code.