**Software Engineering**

**Year 11 , 2025**

**Assessment Task 2**

**Object-Oriented Programming Assignment:**

**“Hunting Wumpus”**

**By: Henry Pham**

Contents

[Software Requirement Specification 3](#_Toc202256247)

[Gantt Chart 6](#_Toc202256248)

[Budget 7](#_Toc202256249)

[Justification of Technology 8](#_Toc202256250)

[Design 10](#_Toc202256251)

[UML Notation 11](#_Toc202256252)

[Class Diagram 12](#_Toc202256253)

[Flowchart 13](#_Toc202256254)

[Pseudocode 14](#_Toc202256255)

[Sequence Diagram 21](#_Toc202256256)

[Graphical User Interface GUI 22](#_Toc202256257)

[Artificial Intelligence Conversion Code 23](#_Toc202256258)

[Explanation of why Java is safer than Python or vice versa 24](#_Toc202256259)

[Security 24](#_Toc202256260)

[Compiling and Execution 24](#_Toc202256261)

[Storing data 26](#_Toc202256262)

[Encryption. 27](#_Toc202256263)

[Why prototyping might be done in Python rather than Java. 28](#_Toc202256264)

[What Tools were used in the development of this Project and their justification 29](#_Toc202256265)

[Visual Code IDE 31](#_Toc202256266)

[Python Compiler 32](#_Toc202256267)

[Artificial Intelligence Converter 33](#_Toc202256268)

[Java 34](#_Toc202256269)

[Code is commented and following industry standard practices 35](#_Toc202256270)

[Justification of Git and GitHub and their difference 36](#_Toc202256271)

[Frequency of committing Code 37](#_Toc202256272)

[Appendix 1 Python Code 38](#_Toc202256273)

[Readme File for Python 51](#_Toc202256274)

[Appendix 2 Java Code 52](#_Toc202256275)

[Readme File for Java 61](#_Toc202256276)

[Reflection 62](#_Toc202256277)

# Software Requirement Specification

Explain the Game. This can be found in the Programming Journal Attached to the Assignment

**Starting the Game:**

* At the start of the game, a player is put in a random room. Also, the cave is randomly spawned with the Wumpus, a number of bats, bottomless pits, and arrows.
* The viewing interface is in the top-down view with simple shape pictures and a representation of the current room and available exits shown in a window.

**Controls:**

* The direction pad can be used to move the player with the use of an arrow.
* When one presses an arrow key with Shift held down, one will shoot an arrow in that direction.
* When the arrow strikes the Wumpus the player is a winner. The player is defeated in case he/she misses and they run out of arrows.

**Win and lose:**

* Win: Get an arrow and kill the Wumpus with the arrow.
* Lose: Enter the same room as the Wumpus (it eats you).
* Drop in a pit with no end.
* Empty out of arrows and did not kill the Wumpus.

**Game Features:**

* Wumpus: is concealed in a room until he is exposed or met. It can change rooms given the ability of mobile movement.
* Bats: In the event that you fall into a room with bats, they will simply select you at random and carry you into a different room and may end up taking you to a dangerous place.
* Pits: If you fall into a pit, it's game over.
* Arrows: There are arrows scattered in the cave and the more you can find, the more will be your count of arrows.

**Hints and Proximity Indicators:**

* If a Wumpus is located in one of the rooms which is adjacent to the current location, then the walls in it are stained with red blood.
* With the presence of bats, there is the squeaking.
* You will get an air current when there is a hole nearby.
* These hints guide the players on when and where to fire the arrows or act a little bit stealthily.

**User Interface:**

* Current room, vicinity threats, amount of arrows, and the place of the player is displayed on the screen.
* Graphics comprise the picture of a player, bats, and Wumpus, and rooms and exits are drawn with the shapes and colours.

## Gantt Chart

An accurate track record of what you did for the project.

A blue and black grid with white lines

AI-generated content may be incorrect.This can be recorded from Git or GitHub.

## Budget

You are a software engineer charging $60 per hour.

Get the time spent from GitHub and multiply by $60 per hour.

Hours worked: 7 hours a week (including class time)

Weeks worked: 7 Weeks

$60 x 7 hours x 7 weeks

= $2940

## Justification of Technology

Why is Python chosen over other languages such as Java?

People prefer Python to Java as it is more straightforward, more laconic, and easier to read/write. Its language is very clean with syntax approaching the natural language and this makes it particularly attractive to beginners and makes it ideal in rapid "prototyping" which is the process of creating an early version of a product, allowing you to test and refine it before committing to full development.

Compared to Java, it takes more boilerplate code and has a more rigid structure but can be prototyped faster and is very flexible in application: data scientists, machine learning developers, web developers, automation, and scripting systems all use Python commonly. Moreover, Python is a highly community-supported and library-rich language, which is a practical and effective option to apply in numerous programs.

What IDE are you using and Why?

My choice of tools is Visual Studio as it is a multipurpose and rich IDE that has built-in support for many languages, including Python with Python workload. It offers a powerful combination of development environments which includes a robust debugging tool, IntelliSense(smart code completion), version control and consists of a well integrated interface that best fits small and even large scale applications.

What are the advantages over other IDE’s?

Visual Studio Vs IDLE:

* Visual Studio is better equipped with the intelliSense and debugging capabilities; IDLE is just a basic editor and debugger.
* Visual Studio supports integrated version control (Git); IDLE does not.
* Visual Studio is capable of handling huge projects; IDLE is only good in small scripts.

Visual Studio Vs PyCharm:

* Visual Studio supports multiple languages (e.g. C++, C#, Python); PyCharm focuses mainly on Python.
* Visual Studio is easier to integrate across the whole Microsoft ecosystem; PyCharm is better at integrating Python-specific features.
* Visual Studio may be more comfortable to work with developers who are already part of the Microsoft ecosystem.

# 

# Design

Here you need to insert such design elements as: UML Notation, i.e. Class Diagram, Sequence Diagrams, Flowchart and Pseudocode, Context Diagram and Explain, Graphical User interface so that a person who never played Wumpus can understand how to play it. You can model this from draw.io website.

A diagram of a computer program

AI-generated content may be incorrect.UML DIAGRAM:

## UML Notation

Why do we use UML Notation?

The purpose of Unified Modelling Language or UML notation is to provide a standardised way to visually represent the design and structure of software systems.

It enables developers, designers and stakeholders the ability to convey complex ideas clearly by diagrams that depict architecture, components and behaviour of a system.

Through a shared visual language, UML allows different teams to analyse requirements, design solutions and document the system in a manner that will assist in the development, and the subsequent maintenance. UML notation improved understanding, coordination, and efficiency throughout the software development process.

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

## A diagram of a flowchart AI-generated content may be incorrect.Flowchart

## Pseudocode

GAME SETUP

----------

DEFINE CONSTANTS:

SCREEN\_WIDTH ← 1000

SCREEN\_HEIGHT ← 1000

COLORS:

BROWN ← (193,154,107)

BLACK ← (0,0,0)

RED ← (138,7,7)

DIRECTIONS:

UP ← 0

DOWN ← 1

LEFT ← 2

RIGHT ← 3

GAME SETTINGS:

NUM\_BATS ← 3

NUM\_PITS ← 3

NUM\_ARROWS ← 0

STARTING\_ARROWS ← 1

MOBILE\_WUMPUS ← FALSE

WUMPUS\_MOVE\_CHANCE ← 50

DECLARE VARIABLES:

player\_pos ← 0

wumpus\_pos ← 0

num\_arrows ← STARTING\_ARROWS

bats\_list ← empty list

pits\_list ← empty list

arrows\_list ← empty list

DEFINE cave as a dictionary mapping each room (1 to 20) to its [UP, DOWN, LEFT, RIGHT] exits

LOAD images:

player\_img, bat\_img, wumpus\_img, arrow\_img

INITIALIZE:

game window, font, display caption

FUNCTION: print\_instructions

----------------------------

DISPLAY game instructions:

- Objective: Kill the Wumpus

- Dangers: Pits, bats, and the Wumpus

- Movement: Use arrow keys

- Shooting: Hold SHIFT + arrow key to shoot

WAIT for player to press ENTER to begin

FUNCTION: populate\_cave

-----------------------

SET player\_pos to random room (1 to 20)

CALL place\_wumpus()

FOR i from 1 to NUM\_BATS:

CALL place\_bat()

FOR i from 1 to NUM\_PITS:

CALL place\_pit()

FOR i from 1 to NUM\_ARROWS:

CALL place\_arrow()

FUNCTION: place\_wumpus

----------------------

REPEAT

SET wumpus\_pos to random room

UNTIL wumpus\_pos ≠ player\_pos

FUNCTION: place\_bat

-------------------

REPEAT

SET bat\_pos to random room

UNTIL bat\_pos ≠ player\_pos AND bat\_pos ≠ wumpus\_pos AND NOT in pits\_list AND NOT in bats\_list

ADD bat\_pos to bats\_list

FUNCTION: place\_pit

-------------------

REPEAT

SET pit\_pos to random room

UNTIL pit\_pos ≠ player\_pos AND pit\_pos ≠ wumpus\_pos AND NOT in bats\_list AND NOT in pits\_list

ADD pit\_pos to pits\_list

FUNCTION: place\_arrow

---------------------

REPEAT

SET arrow\_pos to random room

UNTIL arrow\_pos ≠ player\_pos AND NOT in bats\_list AND NOT in pits\_list AND arrow\_pos ≠ wumpus\_pos

ADD arrow\_pos to arrows\_list

FUNCTION: draw\_room(pos, screen)

-------------------------------

CLEAR screen to BLACK

DRAW a large BROWN circle in the center (the room)

GET exits from cave[pos]

FOR each direction (UP, DOWN, LEFT, RIGHT):

IF an exit exists in that direction:

DRAW a tunnel (rectangle) in that direction

IF wumpus is in a neighboring room:

DRAW RED blood circle

IF current room has a pit:

DRAW BLACK circle

DRAW player at center of screen

IF current room has bat:

DRAW bat image

IF current room has wumpus:

DRAW wumpus image

DISPLAY:

- Player position

- Number of arrows

- Nearby bat or pit warnings (if applicable)

FUNCTION: check\_room(pos)

-------------------------

IF player\_pos == wumpus\_pos:

CALL game\_over("You were eaten by a WUMPUS!")

IF player\_pos in pits\_list:

CALL game\_over("You fell into a bottomless pit!")

IF player\_pos in bats\_list:

DISPLAY "Bats pick you up and drop you somewhere!"

WAIT 2.5 seconds

MOVE bat to new valid random room (not occupied)

MOVE player to new valid random room (not occupied)

IF player\_pos in arrows\_list:

DISPLAY "You found an arrow!"

INCREMENT num\_arrows

REMOVE arrow from arrows\_list

WAIT 2.5 seconds

FUNCTION: move\_wumpus

---------------------

IF MOBILE\_WUMPUS is FALSE OR random chance > WUMPUS\_MOVE\_CHANCE:

RETURN

GET list of exits from current wumpus room

FOR each exit:

IF exit is valid (≠ player AND ≠ bats AND ≠ pits):

MOVE wumpus to that room

BREAK

FUNCTION: shoot\_arrow(direction)

-------------------------------

IF num\_arrows == 0:

RETURN

DECREMENT num\_arrows

IF cave[player\_pos][direction] == wumpus\_pos:

CALL game\_over("You killed the Wumpus!")

ELSE:

DISPLAY "Your arrow disappears into the darkness..."

CALL place\_wumpus()

IF num\_arrows == 0:

CALL game\_over("You are out of arrows. You died.")

FUNCTION: check\_pygame\_events

-----------------------------

GET next user input

IF quit or ESC:

EXIT game

IF key pressed:

IF SHIFT + arrow key:

CALL shoot\_arrow(direction)

ELSE IF movement in that direction is valid:

MOVE player to that connected room

CALL move\_wumpus()

FUNCTION: check\_neighbor\_rooms(pos, item\_list)

---------------------------------------------

FOR each direction in cave[pos]:

IF that room is in item\_list:

RETURN TRUE

RETURN FALSE

FUNCTION: reset\_game

--------------------

CALL populate\_cave()

SET num\_arrows ← STARTING\_ARROWS

FUNCTION: game\_over(message)

----------------------------

WAIT 1 second

FILL screen with RED

DISPLAY message

WAIT 2.5 seconds

PRINT message to console

EXIT game

MAIN GAME LOOP

--------------

CALL print\_instructions()

WAIT for player to press ENTER

INITIALIZE game system and screen

CALL reset\_game()

WHILE game is running:

CALL check\_pygame\_events()

CALL draw\_room(player\_pos, screen)

UPDATE display

CALL check\_room(player\_pos)

## Sequence Diagram

A screen shot of a computer

AI-generated content may be incorrect.

## Graphical User Interface GUI

🎮 Graphical User Interface (GUI) Description

The GUI of *Hunt the Wumpus* is built using the Pygame library. It is a 2D visual representation of a single cave room and includes the following elements:

🕳️ **Room Display**

* Each room is shown as a large black circle in the centre of the screen.
* Exits to adjacent rooms are visualised as rectangular corridors (brown rectangles) pointing up, down, left, or right, depending on which directions are available.

👤 **Player**

* The player is represented by a character image, drawn in the centre of the room.
* The image is loaded from "images/player.png" and displayed using screen.blit().

🦇 **Bats**

* If the player enters a room with bats, a bat image ("images/bat.png") is shown in the centre.
* A message is also shown on screen if bats are nearby: *"You hear the squeaking of bats nearby"*.

🕳️ **Pits**

* If there’s a pit in the current room, a black circle appears inside the room.
* If pits are nearby, a message displays: *"You feel a draft nearby"*.

🧟 **Wumpus**

* If the Wumpus is in the same room, its image ("images/wumpus.png") is shown.
* If the Wumpus is in a neighbouring room, a red warning circle is drawn on the screen to indicate bloodstains, implying danger.

🏹 **Arrows & Stats**

* Text is displayed at the top left showing the player's current position and remaining arrows.
* If the player finds an arrow, a message appears: *"You have found an arrow!"*

📝 **Text Rendering**

* Messages and player stats are rendered using pygame.font.Font, ensuring they're clearly visible.

🖱️ **Controls**

* The GUI listens for arrow key inputs to move the player.
* Holding Shift + Arrow Key lets the player shoot an arrow in that direction.
* Pressing ESC or clicking the window's close button exits the game.

## Artificial Intelligence Conversion Code

Explain what artificial intelligence engine you used to convert the Python Code into Java and were you successful and explain how software and hardware can be used in rapid software development.

I translated my Python code, into Java, using artificial intelligence engine of CodeConvert.AI. This application is the implementation of artificial intelligence models that learn the syntax and semantics of several programming languages and conduct intelligent translation of Python code to Java. It was able to deal with majority of the structural conversion which covers functions, control flow, and variable declaration.

Yes, the conversion was successful — the resulting Java code compiled and ran with only minor adjustments needed for Java-specific conventions, GUI integration with Swing, and type declarations. The final program preserved the full functionality of the original Python version.

Regarding quick software writing, the AI-empowered tools such as CodeConvert.AI can make the process of migrating the code (developed under rapid, versatile language such as Python) into production (converted to a more secure and structured language such as Java) faster. There is also software like IDE (e.g., Visual Studio Code, IntelliJ) that can give both intelligent suggestions, debugging, and respond in real time. Hardware, including multi-core processors and bursts of solid-state drives, also accelerates compiling, testing, and rendering, decreasing development cycles and increasing general efficiency in the development, test, and release of software.

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

# Security

### Compiling and Execution

Security

Java is less prone to security threats than Python since it offers security provisions and is more enforced in enacting encapsulation. Java offers strict access control mechanism through access modifiers (private, protected, public) and this mechanism aids in avoiding unwanted access to the important components of the program. Also, bytecode verification and the Security Manager supported in Java can also help to prevent the malicious behaviour at runtime and particularly in networked or distributed applications. The use of its memory management (through the JVM) and exception handling inbuilt curtail the risks of the buffer overflows and memory leaks.

Conversely Python is more lenient. Although naming conventions used to encourage encapsulation, they are not enforced by the interpreter, and thus it is more likely that sensitive data can be changed by accident. Python also does not have inbuilt sandboxing or secure model that makes it undesirable in applications where security protocols are very critical.

Building and Running

Java is a compiled or a statically typed language and hence type errors and most logic bugs are checked before the program has a chance to run. Such early processing of errors creates a higher degree of reliability and safety. java code compiles into bytecode and it is executed in the Java Virtual Machine (JVM) -- and another level of abstraction is added between the code to isolate it further against the system hardware.

Python is dynamically typed and interpreted, however, ren it is possible to encounter type errors only at the runtime. This increases the possibility of more inconsistency in behaviour and debugging harder in bigger applications. Interpreted execution nature of python also implies execution on line-by-line basis which is useful in development process but not secure in terms of efficient deployment in large or critical systems.

## Storing data

Java has strong and organized ways of data storage, especially as type safe, data integrity and long-term storage applications are required. Java has APIs such as FileWriter, BufferedWriter, ObjectOutputStream and JDBC (Java Database Connectivity) to write it to files or databases. These APIs are also strongly typed and therefore the data format should conform to the declared types hence would help in preventing corruption or misuse. Moreover, in Java, exception handling is in place and thus the writing to the data is not missed.

There is also the flexibility of storing data in python in that, open (), may be used to write to a file, pickle may be used to serialize an object and simple standard database libraries like sqlite3 may be used. The storage facilities of Python are shorter and quicker to apply, and this is good in rapid prototyping. Nevertheless, Python is dynamically typed, which enhances the dangers of types-related anticipated errors on the way of saving and loading the data.

## Encryption.

I did not have to directly encrypt anything in my project because no personal or sensitive information about the user was saved. But in case encryption is to be basically used in a Python based game such as Hunt the Wumpus; the cryptography library can then be suggested as the preferred method of encryption. It is a 3rd party add-on and offers encryption in symmetric forms (e.g. AES), hashing (e.g. the CRC32 or SHA-256), but packages could also encrypt stored save games, player advancement or credentials in case required.

Even though Python is highly flexible and fast to develop, its standard library has little encryption tools, beyond basic hashing (such as hashlib). Programmers are required to add additional libraries and keys and algorithms must be handled by the programmer manually, which is potentially dangerous and requires care.

Comparison Java can offer more secure encryption as default packaged by its official javax.crypto and java.security packages, which are also well-supported and highly integrated into Java ecosystem.

In general, Python can be used in prototyping and performing encryption but all the provision of performing encryption in Python depends on the developer to configure it properly as opposed to Java as it is, which has more robust and enterprise level security built in.

## Why prototyping might be done in Python rather than Java.

Prototyping Python is frequently carried out since it is a dynamically typed, interpreted, and high-level language in which the developers can write and test the ideas fast in an almost theoretical environment with only minimum setup. The syntax of Python is simple, and much closer to the natural language of the person, meaning less amount of codes is required in the Python language to accomplish a certain level of functionality as opposed to Java. This is why Python is a great tool to use in quickly developing an application, experimenting and proof of concept.

Python also has a huge standard library and has third party modules which makes development faster as there is no necessity to implement something, manually. The python language is interpreted, allowing the changes made on the code to be run directly without compilation, lessening the time taken on iteration.

In contrast, Java has more boilerplate code, a compile stage, more type declaration requirements, which developers find compose slower when developing early stages. Java is nonetheless less adaptable and economical in the fast creation and revision of early versions of applications, in spite of the fact that it is more secure and organized to production systems.

This makes Python popular in prototyping, algorithm testing and initial game logic, then converting the application in a compiled language such as Java to perform better, be more secure and scalable in the finished product.

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

PyCharm

The game written in Python was written and debugged using PyCharm. It gives a smart Python development IDE having capabilities such as code defenses, auto-completion, and graphical debugging that integrate development efficient and easy to handle.

Visual Studio Code (VS Code)

VS Code was an editor using little power, which reads and tests Python and Java code quickly. Its included terminal, Git integration, and extension marketplace ensured that it became the perfect choice of code editing flexibility across languages.

IntelliJ IDEA

The Java project was developed and improved through the IntelliJ IDEA. It had an excellent Java development support with intelligent refactoring, powerful debugging, and exceptional error correction all of which helped make the translated code reliable and well sorted.

visual studio IDE

Visual Studio IDE was also investigated to compare it and other development methodologies in particular, debugging capabilities and multi-language projects. It assisted in establishing how various IDEs assist in the large-scale software development processes.

CodeConvert.AI

The AI-based tool which was employed to convert the original Python into Java was CodeConvert.AI (https://www.codeconvert.ai). The tool enabled the initial translation of logic at a fast speed thereby saving time and making the process of rewriting the game in another language easier.

draw.io

The diagrams like UML Class Diagram and Flowchart were drawn by the utilisation of draw.io. It offered an interface that used drag and drop graphics to create the transparent visual representations of the program structure and logic that assisted in documenting its software design professionally.

## Visual Code IDE

The code editor utilised during the completion of this project was the Visual Studio Code (VS Code), which is flexible and efficient code editor. It is an open-source(a type of software developed with no license or fee required), lightweight Integrated Development Environment (IDE) created by Microsoft; the environment supports many programming languages such as Python and Java.

VS Code was especially useful in testing small pieces of code, documentation, and version controlling that works with GitHub. It was especially suited to editing and debugging Python and Java code due to its functionality such as IntelliSense (smart code suggestions), real-time syntax highlighting, error detection and built-in terminal.

Also, the functionality of extensions of VS Code (e.g., Python, Java Pack, Pylance, GitLens) helped me customise the environment to the requirements of the project. This was what made VS Code a multifunctional tool that complemented the more specialised IDEs (such as PyCharm for Python and IntelliJ IDEA for Java) employed at other stages of development.

To conclude, VS Code was not only a stable, lightweight, and customisable editor but also this tool was beneficial in increasing productivity in the work of developing and documentation.

## Python Compiler

I worked with the CPython interpreter that is the most popular and default Python compiler in this project. Though not conventionally compiled as such, Python is technically interpreted, in that the Python Virtual Machine (PVM) executes code in the form of bytecode; CPython compiles Python code (.py files) into bytecode.

The python compiler enabled me to make changes to the Hunt the Wumpus game and have it picked-up and executed within seconds without the use of a complicated build mechanism. This helped the development be quicker and more iterate, and it is perfectly suitable to a project that has continuous testing, game logic changes, and bugs.

The ease with which I needed to run the program with the use of Python which allowed me to write, save and execute immediately worked to my advantage as I was helped to debug quickly. That was especially significant in a game like a real-time one that was developed with Pygame and which requires constant testing of the gameplay and logic elements.

## Artificial Intelligence Converter

I relied on CodeConvert.AI as an online tool to translate the game code aimed at transitioning the implementation language to Java. The tool applies artificial intelligence to understand structure, logical and syntax of Python source code to convert it automatically to functional Java counterparts.

Examples of the most helpful uses of the AI converter were quick translation of languages, and processing of the most simple kinds of functions, conditionals, loops, and data structures. This saved me a lot of time on how I would have written the code manually and enabled me to work on the output to pass the syntax rules of Java and the object orientation concepts.

Even though the AI gave a good base, registration, I still had to do the manual intervention, particularly in dealing with specifics to Java including type declaration, class design, method definition and incorporation with GUI like swings.

All in all, I can say that CodeConvert.AI did a good job in assisting me in narrowing the divide between my Python prototype and a functional Java implementation. It shows how AI can help developers in the development of software in multiple languages and make the software engineering process much faster.

## Java

The Hunt the Wumpus Java implementation was made by translating the original implementation in Python to Java through a manual refactoring with some help of an Artificial Intelligence code translator. This conversion was aimed at investigating the potential advantage of the object-oriented structure and the stronger type system of Java in the development of software especially the secure and robust ones.

Description and Structure

The Java version employs Java Swing to develop a graphical user interface that resembles layout and behaviour of the original Python (Pygame) one. Drawing, rendering and key events are dealt with by a subclass of JPanel and the window, atop which the application runs, is created and managed by a JFrame. Rooms, bats, pits, arrows, and the Wumpus are present in the game; and these all are stored in hash maps and lists to portray state.

Key Java Features implemented:

* Strong typing is a method that assures of data integrity and minimizes runtime errors.
* The implementation of the encapsulation is realized with the help of the private fields and methods.
* Graphics are loaded and displayed by using BufferedImage.
* There is usage of AudioSystem to create sounds upon interactions (e.g. Wumpus encounter, bats, picking up arrows).
* Game loop, redraw by event-driven programming, and paintComponent() the update in response to game state.

Safety and Security Improvements in Java:

* Checking at compile-time ensures that several logic errors, as well as syntax errors, are indicated before the program is executed.
* Encapsulation of data and controlling interaction between components can be achieved by means of access modifiers (private, public).
* Error handling is an inherent feature of the language that encouraged proper error handling.
* All variables and fields are immutable, which is more enforceable in Java, and thread-safe, which is safer with regard to running the game on multiple threads or even over networks.

## Code is commented and following industry standard practices

While building both the Python and Java versions of *Hunt the Wumpus*, I made sure to follow good coding practices so the project would be clean, easy to read, and easy to work with. I added comments throughout the code to explain what each part does—whether it’s placing the Wumpus, drawing the game screen, or checking for danger in nearby rooms. This makes the code much easier to understand, especially if someone else wants to read or change it later.

I used clear, descriptive names for all my variables and functions. Everything is spaced and indented properly, which helps the structure feel organised and professional. In Java, I used try-catch blocks to handle errors like missing image files, and I made sure both games could reset properly after ending, without crashing.

During development, I tested the game by using print statements and playing through different scenarios to make sure everything worked as expected. I also wrote a README file for both versions to help anyone who wants to run the game themselves. Throughout the project, I used GitHub to save my progress regularly, which kept everything backed up and helped me track changes over time. All of this came together to create a final product that’s not just functional, but also polished and built the way real software developers work.

## Justification of Git and GitHub and their difference

Git and GitHub were essential tools in the development of my project. Git is a version control system that tracks changes in code, allowing me to go back to earlier versions if something breaks. It helped me manage my project as it grew more complex. GitHub, on the other hand, is a cloud-based platform where I stored my Git repository online. This made it easy to back up my work, share it, and even access it from different computers. Using Git and GitHub together meant I could track my progress, document my changes, and develop more like a professional software engineer. It also made submitting the project cleaner and more organised.

## Frequency of committing Code

Throughout the development of *Hunt the Wumpus*, I committed my code regularly to GitHub. I made commits after completing major features, like player movement, arrow shooting, and room drawing, as well as after fixing bugs or refining code. This helped me keep track of my progress and made it easier to undo mistakes if something went wrong. By committing often, I followed good software development practices and kept a clear record of how the project evolved over time. It also helped show my learning process and how the game improved step by step.

# Appendix 1 Python Code

Paste your Python Code in Here (White Theme and coloured Syntax)

import pygame

import random

import time

import sys

#===============================================================================

# Functions Area =

#===============================================================================

def check\_neighbor\_rooms(pos, item\_list):

""" Checks each orthagonal cell next to pos for the requested item

returns True as soon as the item is found.

"""

exits = cave[pos]

return any(item in cave[pos] for item in item\_list)

def draw\_room( pos, screen):

""" Draws the room in the back buffer

"""

x=0

y=1

exits = cave[player\_pos]

screen.fill( (0,0,0) ) #paint the background in black

#draw the room circle in brown

circle\_radius = int ((SCREEN\_WIDTH//2)\*.75)

pygame.draw.circle(screen, BROWN, (SCREEN\_WIDTH//2, SCREEN\_HEIGHT//2), circle\_radius, 0)

#next draw all exits from the room

if exits[LEFT] > 0:

left = 0

top = SCREEN\_HEIGHT//2-40

pygame.draw.rect(screen, BROWN, ( (left,top), (SCREEN\_WIDTH//4,80)), 0)

if exits[RIGHT] > 0:

#draw right exit

left = SCREEN\_WIDTH-(SCREEN\_WIDTH//4)

top = SCREEN\_HEIGHT//2-40

pygame.draw.rect(screen, BROWN, ((left,top), (SCREEN\_WIDTH//4,80)), 0)

if exits[UP] > 0:

#draw top exit

left = SCREEN\_WIDTH//2-40

top = 0

pygame.draw.rect(screen, BROWN, ((left,top), (80,SCREEN\_HEIGHT//4)), 0)

if exits[DOWN] > 0 :

#draw bottom exit

left = SCREEN\_WIDTH//2-40

top = SCREEN\_HEIGHT-(SCREEN\_WIDTH//4)

pygame.draw.rect(screen, BROWN, ((left,top), (80,SCREEN\_HEIGHT//4)), 0)

#find out if bats, pits or a wumpus is near

bats\_near = check\_neighbor\_rooms(player\_pos, bats\_list)

pit\_near = check\_neighbor\_rooms(player\_pos, pits\_list)

wumpus\_near = check\_neighbor\_rooms(player\_pos, [wumpus\_pos, [-1,-1]])

#draw a blood circle if the Wumpus is nearby

if wumpus\_near == True:

circle\_radius = int ((SCREEN\_WIDTH//2)\*.5)

pygame.draw.circle(screen, RED, (SCREEN\_WIDTH//2, SCREEN\_HEIGHT//2), circle\_radius, 0)

#draw the pit in black if it is present

if player\_pos in pits\_list:

circle\_radius = int ((SCREEN\_WIDTH//2)\*.5)

pygame.draw.circle(screen, BLACK, (SCREEN\_WIDTH//2, SCREEN\_HEIGHT//2), circle\_radius, 0)

#draw the player

screen.blit(player\_img,(SCREEN\_WIDTH//2-player\_img.get\_width()//2,SCREEN\_HEIGHT//2-player\_img.get\_height()//2))

#draw the bat imag

if player\_pos in bats\_list:

screen.blit(bat\_img,(SCREEN\_WIDTH//2-bat\_img.get\_width()//2,SCREEN\_HEIGHT//2-bat\_img.get\_height()//2))

#draw the wumpus

if player\_pos == wumpus\_pos:

screen.blit(wumpus\_img,(SCREEN\_WIDTH//2-wumpus\_img.get\_width()//2,SCREEN\_HEIGHT//2-wumpus\_img.get\_height()//2))

#draw text

y\_text\_pos = 0 #keeps track of the next y positiojn on screen to draw text

pos\_text = font.render("POS:"+str(player\_pos), 1, (0, 255, 64))

screen.blit(pos\_text,(0, 0))

arrow\_text = font.render("Arrows: "+str(num\_arrows), 1, (0, 255, 64))

y\_text\_pos = y\_text\_pos+pos\_text.get\_height()+10

screen.blit(arrow\_text,(0, y\_text\_pos))

if bats\_near == True:

bat\_text = font.render("You hear the squeaking of bats nearby", 1, (0, 255, 64))

y\_text\_pos = y\_text\_pos+bat\_text.get\_height()+10

screen.blit(bat\_text,(0, y\_text\_pos))

if pit\_near == True:

pit\_text = font.render("You feel a draft nearby", 1, (0, 255, 64))

y\_text\_pos = y\_text\_pos+pit\_text.get\_height()+10

screen.blit(pit\_text,(0, y\_text\_pos))

if player\_pos in bats\_list: #if bats are here, go ahead and flip the display and wait a bit

pygame.display.flip()

time.sleep(2.0)

def populate\_cave():

global player\_pos, wumpus\_pos

#place the player

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

# place the wumpus

place\_wumpus()

#place the bats

for bat in range(0,NUM\_BATS):

place\_bat()

#place the pits

for pit in range (0,NUM\_PITS):

place\_pit()

#place the arrows

for arrow in range (0,NUM\_ARROWS):

place\_arrow()

print ("Player at: "+str(player\_pos))

print ("Wumpus at: "+str(wumpus\_pos))

print ("Bats at:" + str(bats\_list) )

print ("Pits at:" + str(pits\_list))

print ("Arrows at:" +str(arrows\_list))

def place\_wumpus():

global player\_pos, wumpus\_pos

wumpus\_pos = player\_pos

while (wumpus\_pos == player\_pos):

wumpus\_pos = random.randint(0,20)

def place\_bat():

#place the bats

bat\_pos = player\_pos

while bat\_pos == player\_pos or (bat\_pos in bats\_list) or (bat\_pos == wumpus\_pos) or (bat\_pos in pits\_list):

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

bats\_list.append(bat\_pos)

def place\_pit():

pit\_pos = player\_pos

while (pit\_pos == player\_pos) or (pit\_pos in bats\_list) or (pit\_pos == wumpus\_pos) or (pit\_pos in pits\_list):

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

pits\_list.append(pit\_pos)

def place\_arrow():

arrow\_pos = player\_pos

while (arrow\_pos == player\_pos) or (arrow\_pos in bats\_list) or (arrow\_pos == wumpus\_pos) or (arrow\_pos in pits\_list):

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

arrows\_list.append(arrow\_pos)

def check\_room(pos):

global player\_pos, screen, num\_arrows

#is there a Wumpus in the room?

if player\_pos == wumpus\_pos:

game\_over("You were eaten by a WUMPUS!!!")

#is there a pit?

if player\_pos in pits\_list:

game\_over("You fell into a bottomless pit!!")

#is there bats in the room? If so move the player and the bats

if player\_pos in bats\_list:

print("Bats pick you up and place you elsewhere in the cave!")

screen.fill(BLACK)

bat\_text = font.render("Bats pick you up and place you elsewhere in the cave!", 1, (0, 255, 64))

textrect = bat\_text.get\_rect()

textrect.centerx = screen.get\_rect().centerx

textrect.centery = screen.get\_rect().centery

screen.blit(bat\_text,textrect)

pygame.display.flip()

time.sleep(2.5)

#move the bats

new\_pos = player\_pos

while (new\_pos == player\_pos) or (new\_pos in bats\_list) or (new\_pos == wumpus\_pos) or (new\_pos in pits\_list):

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

bats\_list.remove(player\_pos)

bats\_list.append(new\_pos)

print ("bat at: "+str(new\_pos))

#now move the player

new\_pos = player\_pos # set new\_pos equal to the old os so the first test fails

# Now place the player in a random location

while (new\_pos == player\_pos) or (new\_pos in bats\_list) or (new\_pos == wumpus\_pos) or (new\_pos in pits\_list):

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

player\_pos = new\_pos

print ("player at:"+str(player\_pos))

#is there an arrow in the room?

if player\_pos in arrows\_list:

screen.fill(BLACK)

text = font.render("You have found an arrow!", 1, (0, 255, 64))

textrect = text.get\_rect()

textrect.centerx = screen.get\_rect().centerx

textrect.centery = screen.get\_rect().centery

screen.blit(text,textrect)

pygame.display.flip()

time.sleep(2.5)

num\_arrows +=1

arrows\_list.remove(player\_pos)

def reset\_game():

global num\_arrows

populate\_cave()

num\_arrows = 1

def game\_over(message):

global screen

time.sleep(1.0)

screen.fill(RED)

text=font.render(message, 1, (0, 255, 64))

textrect = text.get\_rect()

textrect.centerx = screen.get\_rect().centerx

textrect.centery = screen.get\_rect().centery

screen.blit(text,textrect)

pygame.display.flip()

time.sleep(2.5)

print (message)

pygame.quit()

sys.exit()

def move\_wumpus():

global wumpus\_pos

if mobile\_wumpus == False or random.randint(1,100) > wumpus\_move\_chance:

return

exits = cave[wumpus\_pos]

for new\_room in exits:

if new\_room == 0:

continue

elif new\_room == player\_pos:

continue

elif new\_room in bats\_list:

continue

elif new\_room in pits\_list:

continue

else:

wumpus\_pos = new\_room

break

print ("Wumpus moved to:"+str(wumpus\_pos))

def shoot\_arrow(direction):

global num\_arrows, player\_pos

hit = False

if num\_arrows == 0:

return False

num\_arrows -= 1

if wumpus\_pos == cave[player\_pos][direction]:

hit = True

if hit == True:

game\_over("Your aim was true and you have killed the Wumpus!")

pygame.quit()

sys.exit()

else:

print ("Your arrow sails into the darkness, never to be seen again....")

place\_wumpus()

if num\_arrows == 0:

game\_over("You are out of arrows. You have died!")

pygame.quit()

sys.exit()

def check\_pygame\_events():

global player\_pos

event = pygame.event.poll()

if event.type == pygame.QUIT:

pygame.quit()

sys.exit()

elif event.type == pygame.KEYDOWN:

if event.key == pygame.K\_ESCAPE:

pygame.quit()

sys.exit()

elif event.key ==pygame.K\_LEFT:

if pygame.key.get\_mods() & pygame.KMOD\_SHIFT:

shoot\_arrow(LEFT)

elif cave[player\_pos][LEFT] > 0:

player\_pos=cave[player\_pos][LEFT]

move\_wumpus()

elif event.key == pygame.K\_RIGHT:

if pygame.key.get\_mods() & pygame.KMOD\_SHIFT:

shoot\_arrow(RIGHT)

elif cave[player\_pos][RIGHT] >0:

player\_pos = cave[player\_pos][RIGHT]

move\_wumpus()

elif event.key == pygame.K\_UP:

if pygame.key.get\_mods() & pygame.KMOD\_SHIFT:

shoot\_arrow(UP)

elif cave[player\_pos][UP] > 0:

player\_pos = cave[player\_pos][UP]

move\_wumpus()

elif event.key ==pygame.K\_DOWN:

if pygame.key.get\_mods() & pygame.KMOD\_SHIFT:

shoot\_arrow(DOWN)

elif cave[player\_pos][DOWN] > 0:

player\_pos = cave[player\_pos][DOWN]

move\_wumpus()

def print\_instructoions():

print(

'''

Hunt The Wumpus!

This is the game of "Hunt the Wumpus". You have been cast into a

dark 20 room cave with a fearsome Wumpus. The cave is shaped like a

dodachedron and the only way out is to kill the Wumpus. To that end

you have a bow with one arrow. You might find more arrows from unlucky

past Wumpus victims in the cave. There are other dangers in the cave,

specifcally bats and bottomless pits.

\* If you run out of arrows you die.

\* If you end up in the same room with the Wumpus you die.

\* If you fall into a bottomless pit you die.

\* If you end up in a room with bats they will pick you up

and deposit you in a random location.

If you are near the Wumpus you will see the bloodstains on the walls.

If you are near bats you will hear them and if you are near a bottomless

pit you will feel the air flowing down it.

Use the arrow keys to move. Press the <SHIFT> key and an arrow key to

fire your arrow.

'''

)

#===============================================================================

# Gloabls and Constants area =

#===============================================================================

#Our screen width and height

SCREEN\_WIDTH = SCREEN\_HEIGHT= 1000

#number of bats, pits and arrows in the cave#load our three images

bat\_img = pygame.image.load('images/bat.png')

player\_img = pygame.image.load('images/player.png')

wumpus\_img = pygame.image.load('images/wumpus.png')

arrow\_img = pygame.image.load('images/arrow.png')

#increase the number of bats and pits to make it harder

#increase the number of arrows to make it easier

NUM\_BATS = 3

NUM\_PITS = 3

NUM\_ARROWS = 0

player\_pos = 0 #tracks where we are in the cave

wumpus\_pos = 0 #tracks where the Wumpus is

num\_arrows = 1 # Starting arrows

mobile\_wumpus = False #Set this to true to allow the wumpus to move

wumpus\_move\_chance = 50

#constants for directions

UP = 0

DOWN = 1

LEFT = 2

RIGHT = 3

#color defintions

BROWN = 193,154,107

BLACK = 0,0,0

RED = 138,7,7

cave = {1: [0,8,2,5], 2: [0,10,3,1], 3: [0,12,4,2], 4: [0,14,5,3],

5:[0,6,1,4], 6: [5,0,7,15], 7: [0,17,8,6], 8: [1,0,9,7],

9: [0,18,10,8], 10: [2,0,11,9], 11: [0,19,12,10], 12: [3,0,13,11],

13: [0,20,14,12], 14: [4,0,15,13], 15: [0,16,6,14], 16: [15,0,17,20],

17: [7,0,18,16], 18: [9,0,19,17], 19: [11,0,20,18], 20: [13,0,16,19] }

bats\_list = []

pits\_list = []

arrows\_list = []

#===============================================================================

# Initilizations area =

#===============================================================================

print\_instructoions()

input("Press <ENTER> to begin.")

pygame.init()

screen = pygame.display.set\_mode( (SCREEN\_WIDTH, SCREEN\_HEIGHT), pygame.DOUBLEBUF | pygame.HWSURFACE )

pygame.display.set\_caption("Hunt the Wumpus")

#load our three images

bat\_img = pygame.image.load('images/bat.png')

player\_img = pygame.image.load('images/player.png')

wumpus\_img = pygame.image.load('images/wumpus.png')

arrow\_img = pygame.image.load('images/arrow.png')

#setup our font

font = pygame.font.Font(None, 36)

#Get iniital game settings

reset\_game()

#===============================================================================

# Main Game Loop =

#===============================================================================

while True:

check\_pygame\_events()

draw\_room(player\_pos, screen)

pygame.display.flip()

check\_room(player\_pos)



## Readme File for Python

How do I start the Project

To start the Python version of *Hunt the Wumpus*, make sure you have Python installed on your computer—preferably version 3.8 or above. You'll also need the pygame library, which you can install by opening a terminal or command prompt and running:

pip install pygame

Once installed, make sure all game image files (e.g. bat.png, player.png, wumpus.png, arrow.png) are inside a folder named images, located in the same directory as the Python script. Then, open the Python file (e.g. hunt\_the\_wumpus.py) in your IDE or text editor.

To run the game, simply execute the script by pressing **Run** in your IDE and pressing enter in the terminal.

A game window will open. Use the **arrow keys** to move, and **hold Shift + arrow key** to shoot arrows. The goal is to find and kill the Wumpus without falling into pits or running out of arrows.

If you want to restart the game, close the window and rerun the script.

# Appendix 2 Java Code

Paste your Java Code in here. (White Theme and coloured Syntax).

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.util.\*;

import java.awt.image.\*;

import javax.imageio.ImageIO;

import javax.sound.sampled.\*;

import java.io.\*;

public class HuntTheWumpus extends JPanel implements KeyListener {

private static final int SCREEN\_WIDTH = 1000;

private static final int SCREEN\_HEIGHT = 1000;

private static final int NUM\_BATS = 3;

private static final int NUM\_PITS = 3;

private static final int NUM\_ARROWS = 0;

private static final int UP = 0;

private static final int DOWN = 1;

private static final int LEFT = 2;

private static final int RIGHT = 3;

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

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

private int playerPos = 0;

private int wumpusPos = 0;

private int numArrows = 1;

private boolean mobileWumpus = true;

private int wumpusMoveChance = 50;

private boolean gameOver = false;

private String gameMessage = "";

private BufferedImage playerImg, wumpusImg, batImg, arrowImg, pitImg;

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

private java.util.List<Integer> batsList = new ArrayList<>();

private java.util.List<Integer> pitsList = new ArrayList<>();

private java.util.List<Integer> arrowsList = new ArrayList<>();

public HuntTheWumpus() {

setPreferredSize(new Dimension(SCREEN\_WIDTH, SCREEN\_HEIGHT));

setFocusable(true);

addKeyListener(this);

initCave();

loadImages();

resetGame();

}

private void initCave() {

int[][] data = {

{0,8,2,5},{0,10,3,1},{0,12,4,2},{0,14,5,3},{0,6,1,4},

{5,0,7,15},{0,17,8,6},{1,0,9,7},{0,18,10,8},{2,0,11,9},

{0,19,12,10},{3,0,13,11},{0,20,14,12},{4,0,15,13},

{0,16,6,14},{15,0,17,20},{7,0,18,16},{9,0,19,17},

{11,0,20,18},{13,0,16,19}

};

for (int i = 0; i < data.length; i++) cave.put(i + 1, data[i]);

}

private BufferedImage loadImage(String path) {

try {

return ImageIO.read(new File(path));

} catch (IOException e) {

System.err.println("Could not load image: " + path);

return null;

}

}

private void loadImages() {

playerImg = loadImage("images/player.png");

wumpusImg = loadImage("images/wumpus.png");

batImg = loadImage("images/bat.png");

arrowImg = loadImage("images/arrow.png");

pitImg = loadImage("images/pit.png");

}

private void playSound(String soundFile) {

try {

File file = new File("sounds/" + soundFile);

if (!file.exists()) {

System.err.println("Missing sound: " + soundFile);

return;

}

Clip clip = AudioSystem.getClip();

AudioInputStream inputStream = AudioSystem.getAudioInputStream(file);

clip.open(inputStream);

clip.start();

} catch (Exception e) {

System.err.println("Error playing sound: " + soundFile + " - " + e.getMessage());

}

}

private void resetGame() {

gameOver = false;

gameMessage = "";

batsList.clear();

pitsList.clear();

arrowsList.clear();

Random rand = new Random();

playerPos = rand.nextInt(20) + 1;

do { wumpusPos = rand.nextInt(20) + 1; } while (wumpusPos == playerPos);

for (int i = 0; i < NUM\_BATS; i++) placeEntity(rand, batsList);

for (int i = 0; i < NUM\_PITS; i++) placeEntity(rand, pitsList);

for (int i = 0; i < NUM\_ARROWS; i++) placeEntity(rand, arrowsList);

numArrows = 1;

repaint();

}

private void placeEntity(Random rand, java.util.List<Integer> list) {

int pos;

do {

pos = rand.nextInt(20) + 1;

} while (pos == playerPos || list.contains(pos) || pos == wumpusPos);

list.add(pos);

}

private void checkRoom() {

if (playerPos == wumpusPos) {

playSound("wumpus.wav");

endGame("You were eaten by the WUMPUS!");

} else if (pitsList.contains(playerPos)) {

playSound("pit.wav");

endGame("You fell into a bottomless pit! Press 'R' to restart.");

} else {

if (batsList.contains(playerPos)) {

playSound("bats.wav");

Random rand = new Random();

batsList.remove((Integer) playerPos);

int newBatPos;

do {

newBatPos = rand.nextInt(20) + 1;

} while (batsList.contains(newBatPos) || newBatPos == wumpusPos || pitsList.contains(newBatPos));

batsList.add(newBatPos);

int newPlayerPos;

do {

newPlayerPos = rand.nextInt(20) + 1;

} while (newPlayerPos == playerPos || newPlayerPos == wumpusPos || pitsList.contains(newPlayerPos));

playerPos = newPlayerPos;

gameMessage = "Bats picked you up and dropped you elsewhere!";

}

if (arrowsList.contains(playerPos)) {

playSound("arrow.wav");

numArrows++;

arrowsList.remove((Integer) playerPos);

gameMessage = "You found an arrow!";

}

}

}

private void endGame(String message) {

gameOver = true;

gameMessage = message;

repaint();

}

private void shootArrow(int direction) {

if (numArrows == 0) return;

numArrows--;

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

if (targetRoom == wumpusPos) {

playSound("victory.wav");

endGame("Your aim was true! You killed the Wumpus!");

} else {

playSound("miss.wav");

Random rand = new Random();

do { wumpusPos = rand.nextInt(20) + 1; } while (wumpusPos == playerPos);

if (numArrows == 0) endGame("Out of arrows. You have died! Press 'R' to restart.");

else gameMessage = "You missed. The Wumpus may have moved...";

}

}

private boolean isNear(java.util.List<Integer> list) {

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

for (int room : exits) if (list.contains(room)) return true;

return false;

}

private boolean isWumpusNear() {

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

for (int room : exits) if (room == wumpusPos) return true;

return false;

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

g.setColor(Color.BLACK);

g.fillRect(0, 0, SCREEN\_WIDTH, SCREEN\_HEIGHT);

g.setColor(BROWN);

g.fillOval(SCREEN\_WIDTH / 4, SCREEN\_HEIGHT / 4, SCREEN\_WIDTH / 2, SCREEN\_HEIGHT / 2);

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

if (exits[LEFT] > 0) g.fillRect(0, SCREEN\_HEIGHT / 2 - 40, SCREEN\_WIDTH / 4, 80);

if (exits[RIGHT] > 0) g.fillRect(SCREEN\_WIDTH - SCREEN\_WIDTH / 4, SCREEN\_HEIGHT / 2 - 40, SCREEN\_WIDTH / 4, 80);

if (exits[UP] > 0) g.fillRect(SCREEN\_WIDTH / 2 - 40, 0, 80, SCREEN\_HEIGHT / 4);

if (exits[DOWN] > 0) g.fillRect(SCREEN\_WIDTH / 2 - 40, SCREEN\_HEIGHT - SCREEN\_HEIGHT / 4, 80, SCREEN\_HEIGHT / 4);

if (pitImg != null && pitsList.contains(playerPos)) {

g.drawImage(pitImg, SCREEN\_WIDTH / 2 - pitImg.getWidth() / 2, SCREEN\_HEIGHT / 2 - pitImg.getHeight() / 2, null);

} else if (batImg != null && batsList.contains(playerPos)) {

g.drawImage(batImg, SCREEN\_WIDTH / 2 - batImg.getWidth() / 2, SCREEN\_HEIGHT / 2 - batImg.getHeight() / 2, null);

} else if (wumpusImg != null && playerPos == wumpusPos) {

g.drawImage(wumpusImg, SCREEN\_WIDTH / 2 - wumpusImg.getWidth() / 2, SCREEN\_HEIGHT / 2 - wumpusImg.getHeight() / 2, null);

}

if (!gameOver && playerImg != null) {

g.drawImage(playerImg, SCREEN\_WIDTH / 2 - playerImg.getWidth() / 2, SCREEN\_HEIGHT / 2 - playerImg.getHeight() / 2, null);

}

g.setColor(Color.GREEN);

g.drawString("Position: " + playerPos + " Arrows: " + numArrows, 10, 20);

int y = 50;

if (!gameOver) {

if (isWumpusNear()) { g.drawString("You see bloodstains on the walls.", 10, y); y += 20; }

if (isNear(batsList)) { g.drawString("You hear the squeaking of bats.", 10, y); y += 20; }

if (isNear(pitsList)) { g.drawString("You feel a draft.", 10, y); y += 20; }

}

if (!gameMessage.isEmpty()) {

g.setColor(RED);

g.drawString(gameMessage, 10, y);

}

if (gameOver) {

g.setFont(new Font("Arial", Font.BOLD, 36));

g.setColor(Color.RED);

g.drawString("GAME OVER", SCREEN\_WIDTH / 2 - 120, SCREEN\_HEIGHT / 2 + 200);

g.setFont(new Font("Arial", Font.PLAIN, 18));

g.drawString("Press 'R' to restart or use Game > Restart menu", SCREEN\_WIDTH / 2 - 200, SCREEN\_HEIGHT / 2 + 240);

}

}

@Override public void keyPressed(KeyEvent e) {

int key = e.getKeyCode();

if (gameOver && key == KeyEvent.VK\_R) {

resetGame();

return;

}

if (gameOver) return;

boolean shift = (e.getModifiersEx() & KeyEvent.SHIFT\_DOWN\_MASK) != 0;

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

if (key == KeyEvent.VK\_LEFT) {

if (shift) shootArrow(LEFT);

else if (exits[LEFT] > 0) playerPos = exits[LEFT];

} else if (key == KeyEvent.VK\_RIGHT) {

if (shift) shootArrow(RIGHT);

else if (exits[RIGHT] > 0) playerPos = exits[RIGHT];

} else if (key == KeyEvent.VK\_UP) {

if (shift) shootArrow(UP);

else if (exits[UP] > 0) playerPos = exits[UP];

} else if (key == KeyEvent.VK\_DOWN) {

if (shift) shootArrow(DOWN);

else if (exits[DOWN] > 0) playerPos = exits[DOWN];

}

checkRoom();

repaint();

}

@Override public void keyReleased(KeyEvent e) {}

@Override public void keyTyped(KeyEvent e) {}

public static void main(String[] args) {

SwingUtilities.invokeLater(() -> {

JFrame frame = new JFrame("Hunt the Wumpus");

HuntTheWumpus gamePanel = new HuntTheWumpus();

JMenuBar menuBar = new JMenuBar();

JMenu gameMenu = new JMenu("Game");

JMenuItem restartItem = new JMenuItem("Restart");

restartItem.addActionListener(e -> gamePanel.resetGame());

gameMenu.add(restartItem);

menuBar.add(gameMenu);

frame.setJMenuBar(menuBar);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setContentPane(gamePanel);

frame.pack();

frame.setLocationRelativeTo(null);

frame.setVisible(true);

});

}

}

## Readme File for Java

How do I start the Project

Requirements:

* Java Development Kit (JDK) 17 or higher
* Java-compatible IDE (e.g. IntelliJ IDEA, Eclipse, or Visual Studio Code with Java extensions)
* Images and sound assets folder:  
  Ensure you have the following in the project root

Running the Game:

1. Open the project folder in your Java IDE.
2. Compile the HuntTheWumpus.java file.
3. Run the main() method found at the bottom of the HuntTheWumpus class. This will:
   * Launch a game window using JFrame
   * Display the game screen
   * Enable arrow key movement and Shift+arrow to shoot

Controls:

* Arrow Keys: Move between rooms.
* Shift + Arrow Key: Shoot an arrow in the direction pressed.
* R Key: Restart the game after game over.
* Game Menu > Restart: Also restarts the game.

Troubleshooting:

* Missing images or sounds: Make sure the /images and /sounds folders are placed in the correct directory and contain the appropriate files.
* Game not launching: Ensure your Java SDK is correctly configured and you're using a Java GUI-supported IDE.

# Reflection

What did you learn from this project?

Throughout this portfolio, I have developed a deeper understanding of programming by working with both Python and Java. The project gave me practical experience in writing clean, well-structured code that follows industry best practices, such as clear naming conventions, modular design, and comprehensive commenting. This made the code easier to read, maintain, and debug.

In the Python portion of the portfolio, I strengthened my problem-solving skills by building functions, handling user input, and managing game logic. I became more comfortable with Python’s dynamic nature and its extensive libraries, which allowed me to implement features efficiently. Debugging and testing also became more intuitive as I learned to trace errors systematically and improve program stability.

The Java section complemented my Python experience by exposing me to a statically typed, object-oriented language. Writing Java code helped me understand the importance of strict data types, method structures, and the use of classes to encapsulate behaviour. It was valuable to see how different languages approach similar problems, broadening my programming mindset and adaptability.

Overall, this portfolio reflects my growth as a programmer—improving technical skills, learning new languages, and applying good software development principles. It has given me confidence to tackle more complex projects in the future and the flexibility to switch between programming languages while maintaining quality and clarity in my code.