**Software Engineering**

**Year 11 , 2025**

**Assessment Task 2**

**Object-Oriented Programming Assignment:**

**“Hunting Wumpus”**

**By: Dylan Van**

**Contents**

[Software Requirement Specification 3](#_Toc198125157)

[Gantt Chart 4](#_Toc198125158)

[Budget 5](#_Toc198125159)

[Justification of Technology 6](#_Toc198125160)

[UML Notation 7](#_Toc198125162)

[Class Diagram 8](#_Toc198125163)

[Flowchart 9](#_Toc198125164)

[Pseudocode 10](#_Toc198125165)

[Sequence Diagram 11](#_Toc198125166)

[Graphical User Interface GUI 12](#_Toc198125167)

[Artificial Intelligence Conversion Code 15](#_Toc198125168)

[Explanation of why Java is safer than Python or vice versa 16](#_Toc198125169)

[Why prototyping might be done in Python rather than Java. 17](#_Toc198125174)

[What Tools were used in the development of this Project and their justification 18](#_Toc198125175)

[Appendix 1 Python Code 20](#_Toc198125183)

[Readme File for Python 33](#_Toc198125184)

[Appendix 2 Java Code 34](#_Toc198125185)

[Readme File for Java 39](#_Toc198125186)

[Reflection 40](#_Toc198125187)

# **Software Requirement Specification**

"Hunt the Wumpus" is a classic, text-based role-playing adventure game where the player explores a dangerous cave in search of a fearsome creature known as the Wumpus. The cave consists of 20 interconnected rooms, and the objective is to hunt and kill the Wumpus before falling victim to other cave hazards.

**Objective**

The player's goal is to:

* Survive the cave’s hazards (bats, pits, and the Wumpus).
* Find arrows to use as weapons.
* Kill the Wumpus using those arrows.
* Avoid death at all costs.

**Game Mechanics**

* Players move through the cave using the arrow keys.
* They can shoot arrows in any direction using Shift + Arrow Key.
* If they shoot and hit the Wumpus, they win.
* If they run out of arrows, fall into a pit, or meet the Wumpus unarmed, they lose.

**Hazards**

* Wumpus: If the player enters the Wumpus's room, they are eaten.
* Pits: Falling into a bottomless pit ends the game instantly.
* Bats: If the player enters a bat room, they are picked up and dropped in a random room, possibly into danger.

**Hints**

The game gives subtle hints:

* “You hear squeaking” means bats are nearby.
* “You feel a draft” means there’s a pit nearby.
* “You see bloodstains” means the Wumpus is close.

**Gameplay Features**

* Classic maze-like exploration.
* Random placement of hazards at the start of each game.
* Strategic movement and arrow use.
* Visual elements using pygame for room and character rendering.

## **Gantt Chart**

Throughout the development of the “Hunt the Wumpus” game, I consistently used GitHub to track all progress. Initially, I created a new private repository where I committed the basic game structure including folders for assets, modules, and documentation. Early commits involved setting up the main Python script, defining core classes like Player, Room, and Wumpus, and implementing basic movement logic.

Midway through the project, I committed updates for the game logic, such as arrow shooting, Wumpus behaviour, and cave generation. I used frequent commits with clear messages like “Added arrow shooting mechanic”, “Fixed player movement bug”, and “Refactored cave generation into separate modules” to ensure traceability.

I also committed documentation artifacts, including the UML class diagram, pseudocode, and flowchart files. Git branches were used to experiment with a Graphical User Interface (GUI), though this feature remained a stretch goal.

Toward the end of the timeline, I committed the Java translation of the Python code using an AI code converter, followed by manual debugging and testing commits. Finally, I pushed the complete portfolio including the Python source code, Java version, and documentation into a final commit titled “Final Submission: Python and Java code with documentation”.

This version-controlled history on GitHub provides a reliable, timestamped record of my contributions, enabling clear project tracking and accountability.

## **Budget**

I have spent 12 hours using GitHub to create and edit my repository for Hunt The Wumpus and being charged $60 per hour.

12 x 60 = $720

Therefore, my total budget being charged as a software engineer is $720.

## **Justification of Technology**

Python was chosen for this project primarily because of its simplicity, readability, and rapid development capabilities. Unlike Java, which has a more complex syntax and requires more boilerplate code, Python allows developers to write logic faster and with fewer lines. This makes Python ideal for student projects and rapid prototyping such as “Hunt the Wumpus”.

Additionally, Python has a rich standard library and extensive community support. It also supports multiple programming paradigms, including object-oriented and procedural programming, making it versatile and accessible. Python is interpreted, which means you can run your code immediately after writing it. This is especially useful during development and debugging.

## **UML Notation**

**1. Visual Planning**

UML helps you plan your code structure visually before you write it. You can design:

* Classes (e.g., Player, Wumpus, Cave)
* Relationships between objects
* How the game flows (e.g., sequence of moves or decisions)

**2. Object-Oriented Design**

Since Hunt the Wumpus uses an Object-Oriented Programming (OOP) approach, UML helps you:

* Define inheritance, attributes, and methods
* Organise your classes clearly
* Show how objects interact with each other (via sequence diagrams)

**3. Team Communication**

UML diagrams are easy to read and share, so they help when:

* Discussing code with a teacher or teammates
* Documenting the project for assessment or future development

**4. Error Prevention**

By planning ahead using UML, you can:

* Spot design issues early
* Avoid writing unnecessary or poorly structured code

**Summary:**

We use UML to think before we code. It helps us design, communicate, and debug our programs more effectively.

## **Class Diagram**

A screenshot of a computer program

AI-generated content may be incorrect.

## **Flowchart**

A diagram of a computer program

AI-generated content may be incorrect.

## **Pseudocode**

START GAME

  Initialize rooms, player, and hazards

  WHILE game is running:

    DISPLAY current room and nearby warnings

    PROMPT player to move or shoot

    IF move:

      GET chosen room

      IF valid:

        player moves

        CHECK if Wumpus, Pit, or Bat

      ELSE:

        DISPLAY invalid move

    IF shoot:

      IF player has arrows:

        DECREASE arrow count

        CHECK if Wumpus is in chosen room

        IF hit:

          DISPLAY win

          END game

        ELSE:

          Wumpus may move

      ELSE:

        DISPLAY "no arrows"

  DISPLAY game over

END

## **Sequence Diagram**

A diagram of a game

AI-generated content may be incorrect.

## **Graphical User Interface GUI**

import tkinter as tk

import random

from tkinter import messagebox

class Room:

    def \_\_init\_\_(self, id, connections):

        self.id = id

        self.connections = connections

class WumpusGameGUI:

    def \_\_init\_\_(self, root):

        self.root = root

        self.root.title("Hunt the Wumpus")

        self.rooms = self.create\_rooms()

        self.player\_room = random.choice(self.rooms)

        self.wumpus\_room = random.choice([r for r in self.rooms if r != self.player\_room])

        self.create\_widgets()

        self.update\_ui()

    def create\_rooms(self):

        # Create 5 simple interconnected rooms

        return [

            Room(1, [2, 3]),

            Room(2, [1, 4]),

            Room(3, [1, 5]),

            Room(4, [2]),

            Room(5, [3])

        ]

    def create\_widgets(self):

        self.label = tk.Label(self.root, text="", font=("Helvetica", 14))

        self.label.pack(pady=10)

        self.buttons\_frame = tk.Frame(self.root)

        self.buttons\_frame.pack(pady=10)

    def update\_ui(self):

        self.label.config(text=f"You are in Room {self.player\_room.id}")

        # Clear old buttons

        for widget in self.buttons\_frame.winfo\_children():

            widget.destroy()

        # Create new move buttons

        for room\_id in self.player\_room.connections:

            btn = tk.Button(

                self.buttons\_frame,

                text=f"Move to Room {room\_id}",

                command=lambda rid=room\_id: self.move\_to\_room(rid),

                width=20

            )

            btn.pack(pady=2)

        # Warning if near Wumpus

        if self.wumpus\_room.id in self.player\_room.connections:

            messagebox.showinfo("Warning!", "You smell something terrible nearby...")

    def move\_to\_room(self, room\_id):

        new\_room = next(r for r in self.rooms if r.id == room\_id)

        self.player\_room = new\_room

        if self.player\_room == self.wumpus\_room:

            messagebox.showerror("Game Over", "The Wumpus ate you! Game Over.")

            self.root.destroy()

        else:

            self.update\_ui()

# Main loop

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

    root = tk.Tk()

    game = WumpusGameGUI(root)

    root.mainloop()

## **Artificial Intelligence Conversion Code**

To assist in converting the Hunt the Wumpus Python code into Java, I used an AI code conversion tool powered by OpenAI's ChatGPT. This engine is capable of understanding programming logic and generating equivalent code in another language while maintaining structure and functionality.

The AI engine correctly translated many core components (such as classes, method definitions, and control structures), but I had to manually adjust or rewrite parts of the code due to key differences between Python and Java, such as: Strict typing in Java (e.g., specifying int, String, etc.), Java’s lack of built-in dynamic data structures like Python’s dictionaries, Need for getter/setter methods in Java for encapsulation, Manual handling of input/output in Java. The AI made the initial translation faster, but full success required understanding both languages and making logical adjustments.

AI tools like ChatGPT accelerate development by helping with code translation, debugging, and documentation, but developer knowledge is still essential. Combined with powerful software and hardware, they enable rapid, efficient, and collaborative software development.

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

**Security**

Java is generally considered more secure by default for enterprise or large systems. Python is safe for many applications, but requires careful coding practices due to its flexibility.

**Compiling and Execution**

Java catches more errors before you run the program. Python allows you to write and test faster, but runtime errors are more common if not carefully tested.

**Storing data**

Both can store data well, but Java enforces stricter controls and is better suited for highly secure data storage. Python’s ease of use can sometimes sacrifice safety if developers are careless.

**Encryption**

Java has stronger built-in support for encryption, certificates, and regulatory compliance.

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

Python is often chosen for prototyping over Java because of its simplicity, speed, and flexibility. Python’s syntax is much more concise and readable, allowing developers to write and modify code quickly without the need for strict type declarations or complex setup. Its dynamic typing and interactive interpreter make it easy to test small pieces of code on the fly, which is ideal during the early stages of development when requirements may still be changing. Additionally, Python offers a vast ecosystem of third-party libraries for data analysis, machine learning, GUI design, and web development, enabling developers to rapidly add features without building everything from scratch. In contrast, Java’s strict object-oriented structure, compile-time type checking, and more complex setup processes make it more suitable for production-level software but slower for initial prototyping. As a result, Python is often the preferred language for quickly building and testing early versions of applications, especially when working with AI, data science, or experimental projects.

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

**1. Python Programming Language**

Python was chosen for its simplicity, readability, and rapid development capabilities. Its syntax is clean and easy to learn, which is ideal for educational projects and quick prototyping. Python also has strong community support and extensive libraries, making game development more accessible, especially with frameworks like Pygame.

**2. Pygame Library**

Pygame is a set of Python modules designed for writing video games. It simplifies handling graphics, sounds, and user input. It’s lightweight and well-documented, perfect for building a 2D text-based adventure game with basic graphics, like “Hunt the Wumpus.”

**3. OpenAI ChatGPT**

Used to assist in converting Python code to Java. This AI tool accelerates the process of understanding code logic and generating equivalent code in another programming language. It helped in initial code translation and problem-solving during development.

**4. Git & GitHub**

Git was used for version control to track code changes, manage different versions, and enable collaboration. GitHub provided a remote repository platform to back up the code, facilitate teamwork, and share the project with teachers or peers. It also supports issue tracking and project management.

**5. Integrated Development Environment: Visual Studio Code**

VS Code was used because it supports Python and Java development, has a vast ecosystem of extensions, and offers integrated terminal and Git support. It improves productivity through features like code autocomplete, syntax highlighting, debugging, and error detection. For Java, IntelliJ IDEA is often preferred due to its strong refactoring tools and deep Java support.

**6. Microsoft Word**

Microsoft Word was used to create documentation, including the journal, Gantt Chart, flowcharts, UML diagrams, and other project artifacts. It is widely used, supports rich formatting, and is a standard for professional documentation submissions.

**7. Diagramming Tools**

These tools were used to create UML diagrams, flowcharts, data flow diagrams, and sequence models. They provide intuitive drag-and-drop interfaces and templates that help visualize system architecture and design, which improves understanding and communication.

**8. Time Tracking Tools**

Tools like Toggl or GitHub time tracking features helped estimate development time for budgeting and reporting the cost of the project.

**Summary:**

Each tool was selected to optimise productivity, collaboration, clarity, and code quality throughout the development lifecycle. Together, they created an environment that supported both efficient coding and comprehensive project documentation.

# **Appendix 1 Python Code**

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.

    '''

    )

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

#                       Globals 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 definitions

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 = []

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

#                       Initializations 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**

**Hunt The Wumpus - Python Edition**

**Project Description**

This project is a Python implementation of the classic "Hunt The Wumpus" text-based adventure game. The player explores a dangerous cave full of hazards such as bats and bottomless pits while hunting the Wumpus monster. The game features a simple graphical interface built using the Pygame library.

**How to Start the Project:**

1. Install Python

* Download and install \*\*Python 3.x\*\* from the official website: <https://www.python.org/downloads/>

1. Install Pygame

* Open a terminal or command prompt and install the Pygame library by running: pip install pygame

1. Download or Clone the Repository

* Clone this project from GitHub: git clone <https://github.com/your-username/your-repository-name.git>
* Or download the ZIP file and extract it.

1. Verify Project Files

* Ensure the following files and directories are present: /project-folder, main.py, /images, bat.png, player.png, wumpus.png, arrow.png

1. Run the Game

* In your terminal or command prompt, navigate to the project folder: cd project-folder
* Run the game using: python main.py
* The game window should launch, and you're ready to play!

# **Appendix 2 Java Code**

import java.util.\*;

public class HuntTheWumpus {

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

    private static int playerPos;

    private static int wumpusPos;

    private static Set<Integer> pits = new HashSet<>();

    private static Set<Integer> bats = new HashSet<>();

    private static int arrows = 3;

    private static Random rand = new Random();

    public static void main(String[] args) {

        setupCave();

        populateCave();

        gameLoop();

    }

    private static void setupCave() {

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

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

        cave.put(3, new int[]{12, 4, 2});

        cave.put(4, new int[]{14, 5, 3});

        cave.put(5, new int[]{6, 1, 4});

        cave.put(6, new int[]{5, 7, 15});

        cave.put(7, new int[]{17, 8, 6});

        cave.put(8, new int[]{1, 9, 7});

        cave.put(9, new int[]{18, 10, 8});

        cave.put(10, new int[]{2, 11, 9});

        cave.put(11, new int[]{19, 12, 10});

        cave.put(12, new int[]{3, 13, 11});

        cave.put(13, new int[]{20, 14, 12});

        cave.put(14, new int[]{4, 15, 13});

        cave.put(15, new int[]{6, 16, 14});

        cave.put(16, new int[]{15, 17, 20});

        cave.put(17, new int[]{7, 18, 16});

        cave.put(18, new int[]{9, 19, 17});

        cave.put(19, new int[]{11, 20, 18});

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

    }

    private static void populateCave() {

        playerPos = rand.nextInt(20) + 1;

        do {

            wumpusPos = rand.nextInt(20) + 1;

        } while (wumpusPos == playerPos);

        while (pits.size() < 3) {

            int pos = rand.nextInt(20) + 1;

            if (pos != playerPos && pos != wumpusPos) pits.add(pos);

        }

        while (bats.size() < 3) {

            int pos = rand.nextInt(20) + 1;

            if (pos != playerPos && pos != wumpusPos && !pits.contains(pos)) bats.add(pos);

        }

    }

    private static void gameLoop() {

        Scanner scanner = new Scanner(System.in);

        while (true) {

            System.out.println("You are in room " + playerPos);

            System.out.println("Tunnels lead to: " + Arrays.toString(cave.get(playerPos)));

            senseNearby();

            System.out.print("Move (M) or Shoot (S)? ");

            String action = scanner.next().toUpperCase();

            if (action.equals("M")) {

                move(scanner);

            } else if (action.equals("S")) {

                shoot(scanner);

            } else {

                System.out.println("Invalid command.");

            }

            checkCurrentRoom();

        }

    }

    private static void senseNearby() {

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

            if (neighbor == wumpusPos)

                System.out.println("You smell a terrible stench.");

            if (pits.contains(neighbor))

                System.out.println("You feel a cold wind blowing from a nearby cavern.");

            if (bats.contains(neighbor))

                System.out.println("You hear rustling of bat wings.");

        }

    }

    private static void move(Scanner scanner) {

        System.out.print("Enter room to move to: ");

        int moveTo = scanner.nextInt();

        if (Arrays.stream(cave.get(playerPos)).anyMatch(r -> r == moveTo)) {

            playerPos = moveTo;

        } else {

            System.out.println("You can't move there.");

        }

    }

    private static void shoot(Scanner scanner) {

        if (arrows <= 0) {

            System.out.println("No arrows left!");

            return;

        }

        System.out.print("Enter room to shoot into: ");

        int shootTo = scanner.nextInt();

        arrows--;

        if (shootTo == wumpusPos) {

            System.out.println("You killed the Wumpus! You win!");

            System.exit(0);

        } else {

            System.out.println("You missed...");

            if (rand.nextInt(100) < 75) {

                moveWumpus();

            }

            if (arrows == 0) {

                System.out.println("You have run out of arrows. Game Over.");

                System.exit(0);

            }

        }

    }

    private static void moveWumpus() {

        int[] possibleRooms = cave.get(wumpusPos);

        wumpusPos = possibleRooms[rand.nextInt(possibleRooms.length)];

        System.out.println("The Wumpus has moved!");

    }

    private static void checkCurrentRoom() {

        if (playerPos == wumpusPos) {

            System.out.println("The Wumpus got you! Game Over.");

            System.exit(0);

        }

        if (pits.contains(playerPos)) {

            System.out.println("You fell into a bottomless pit! Game Over.");

            System.exit(0);

        }

        if (bats.contains(playerPos)) {

            System.out.println("Bats carried you to another room!");

            playerPos = rand.nextInt(20) + 1;

            checkCurrentRoom(); // recheck new room

        }

    }

}

## **Readme File for Java**

**Hunt The Wumpus - Java Edition**

**Project Description**

This is a Java-based version of the classic adventure game Hunt The Wumpus. The player explores a dangerous cave while trying to avoid pits and bats, and ultimately kill the Wumpus. The Java version was converted from Python to demonstrate object-oriented principles and improve code safety and structure.

**How to Start the Project**

1. Install Java Development Kit

- Download and install the “Java JDK (version 8 or higher)”:

<https://www.oracle.com/java/technologies/javase-jdk-downloads.html>

1. Install an IDE

- IntelliJ IDEA

<https://www.jetbrains.com/idea/download/>

- Eclipse

<https://www.eclipse.org/downloads/>

- NetBeans

<https://netbeans.apache.org/download/index.html>

1. Download or Clone the Repository

- Clone this project from GitHub:

<https://github.com/your-username/your-repository-name.git>

* Or download the ZIP file and extract it

1. Import the Project into Your IDE

* Open your IDE
* Select "Open Project" or "Import Project" and choose the downloaded project folder.
* If you use an IDE, it will automatically detect and configure the Java project.

1. Compile and Run

* If using terminal: javac Main.java,  java Main
* If using an IDE: Simply click the Run button after importing

# **Reflection**

Working on the "Hunt the Wumpus" game project has been a valuable learning experience that helped me grow both technically and creatively. One of the biggest lessons I learned was how to use Python more effectively, especially by applying object-oriented programming (OOP) principles. Creating classes for different parts of the game, like the player, hazards, and rooms, helped me structure the code in a clear and manageable way. This project gave me a stronger understanding of how important OOP is, not just for games but for any software that needs to be scalable and easy to update.

Another important part of this project was using various software development tools. For example, GitHub allowed me to save versions of my code and track my progress over time. It also prepared me for what it's like to work in real-world development environments where version control is essential. Planning tools such as Gantt charts and flowcharts helped me visualize the overall structure of the game and manage my time better. I also used UML notation to map out class relationships, which made the design phase clearer before jumping into writing code.

One interesting challenge I faced was converting parts of the Python code into Java using an artificial intelligence (AI) code converter. While the tool gave me a good starting point, I learned that automated solutions are not perfect. I had to manually fix syntax errors and restructure the code to make it work in Java’s stricter environment. This taught me to not blindly trust AI tools, but to treat them as assistants rather than complete solutions. It also helped me see why Java is sometimes considered more secure and robust for larger systems, especially when dealing with data.

Finally, this project taught me the importance of software planning, debugging, and adaptability. Whenever bugs came up, I learned to test logically and break the problem down step by step. I also saw how helpful it is to create documentation, both for planning and for explaining how the game works. These practices will definitely help me in future projects, whether in school or in a professional setting. Overall, this project has made me a more confident programmer and problem-solver.