**Software Engineering**

**Year 11, 2025**

**Assessment Task 2**

**Object-Oriented Programming Assignment:**

**“Hunting Wumpus”**

**By: Peter Trinh**

Contents

[Software Requirement Specification 3](#_Toc198125157)

[Gantt Chart 4](#_Toc198125158)

[Budget 5](#_Toc198125159)

[Justification of Technology 6](#_Toc198125160)

[Design 7](#_Toc198125161)

[UML Notation 8](#_Toc198125162)

[Class Diagram 9](#_Toc198125163)

[Flowchart 10](#_Toc198125164)

[Pseudocode 11](#_Toc198125165)

[Sequence Diagram 12](#_Toc198125166)

[Graphical User Interface GUI 13](#_Toc198125167)

[Artificial Intelligence Conversion Code 14](#_Toc198125168)

[Explanation of why Java is safer than Python or vice versa 15](#_Toc198125169)

[Security 15](#_Toc198125170)

[Compiling and Execution 15](#_Toc198125171)

[Storing data 16](#_Toc198125172)

[Encryption. 17](#_Toc198125173)

[Why prototyping might be done in Python rather than Java. 18](#_Toc198125174)

[What Tools were used in the development of this Project and their justification 19](#_Toc198125175)

[Visual Code IDE 20](#_Toc198125176)

[Python Compiler 21](#_Toc198125177)

[Artificial Intelligence Converter 22](#_Toc198125178)

[Java 23](#_Toc198125179)

[Code is commented and following industry standard practices 24](#_Toc198125180)

[Justification of Git and GitHub and their difference 25](#_Toc198125181)

[Frequency of committing Code 26](#_Toc198125182)

[Appendix 1 Python Code 27](#_Toc198125183)

[Readme File for Python 28](#_Toc198125184)

[Appendix 2 Java Code 29](#_Toc198125185)

[Readme File for Java 30](#_Toc198125186)

[Reflection 31](#_Toc198125187)

# Software Requirement Specification

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

*Hunt the Wumpus* is a classic text-based adventure game where the player explores a dark, underground cave system made up of connected rooms. The goal of the game is to hunt and kill the **Wumpus**, a mysterious and dangerous creature hiding somewhere in the cave.

The player can move from room to room, one at a time, and must avoid deadly hazards like:

* **Bottomless pits** – falling into one means instant death.
* **Super bats** – they pick the player up and drop them in a random room, which can be risky.
* **The Wumpus itself** – if the player enters the Wumpus's room, the creature might wake up and eat them.

To defeat the Wumpus, the player must carefully explore the cave, using clues (like smells, breezes, or sounds) to figure out where the hazards are without stepping directly into danger. Once they think they know where the Wumpus is, they can shoot an arrow into a nearby room to try to kill it. But if they miss, the Wumpus might move!

The game mixes logic, strategy, and a bit of luck, and it encourages the player to think critically about their surroundings to survive and win.

## Gantt Chart

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

This can be record from Git or GitHub.

A blue stairs with white text

AI-generated content may be incorrect.

## Budget

You are a software engineer charging $60 per hour.

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

Worked Hours: 4 hrs/ week   
for 7 weeks

🡪7x5= 35 hrs

🡪35x60= $2100

## Justification of Technology

Why is Python chosen over other language such as Java. What IDE are you using and why? What are the advantages over other IDE.

The main reason Python is chosen over Java for developing *Hunt the Wumpus* is its simplicity and ease of use, especially for high school programmers like me. Python has a more readable and straightforward syntax, which makes it easier to understand and work with, even for those who are relatively new to programming. In contrast, Java tends to be more complex, requiring more lines of code and stricter rules, which can make it harder to grasp for beginners.

Although Java does offer faster processing power and greater efficiency in large-scale applications, this advantage isn’t necessary for a game like *Hunt the Wumpus*. The game doesn’t rely on reaction-based mechanics or real-time performance, so Python’s slightly slower speed doesn’t impact the gameplay experience. In this case, the benefits of Python’s simplicity and quick development time outweigh the performance advantages of Java. Therefore, Python is the more suitable choice for this type of project.

# Design

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

## UML Notation

Why do we use UML Notation?

UML (Unified Modelling Language) is a visual diagram language used to represent the structure and design of the software system. This helps developers, designers and stakeholders to concept and express more complex ideas of system architecture, components and behaviour through drawing. A uniform set of symbols and formats makes the UML ideal to help the teams understand what is necessary, design a solution, and document the system for further development and long -term maintenance. UML, in short, software development promotes communication, cooperation and efficiency in the life cycle.

## Class Diagram

A diagram of a room

AI-generated content may be incorrect.

## Flowchart

A screenshot of a computer

AI-generated content may be incorrect.

## 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 screenshot of a computer

AI-generated content may be incorrect.

## 

## Graphical User Interface GUI

**What You See on the Screen**

**The Cave Room**

* When the game starts, you see a **brown circular room** on a **black background**.
* This represents the room you're currently in.
* Around the edges of the circle, you might see **brown rectangular openings** — these are **exits** to other rooms. If there is an opening on a side, you can move in that direction.

**Your Character**

* In the center of the room, there is an **image of a person** — this is **you**, the player.

**Dangers and Clues (Visual and Text Warnings)**

The cave contains hidden dangers:

* A **Wumpus** (a monster)
* **Bats**
* **Bottomless pits**

The game gives you hints when these are **nearby**, even if they're not in the current room:

**1. Wumpus Nearby**

* A **red circle** appears around your player. This is a warning that the Wumpus is very close.

**2. Pit Nearby**

* A message appears on the screen: **“You feel a draft nearby”** — this means a pit is in a neighboring room.

**3. Bats Nearby**

* A message appears: **“You hear the squeaking of bats nearby”** — this means bats are close by.

**What Happens When You Enter a Room?**

The game checks the contents of the room you move into:

* **If the Wumpus is there**: You are eaten, and the game ends.
* **If there’s a pit**: You fall and die.
* **If there are bats**: They pick you up and drop you in a **random room**. They also move to a new spot.
* **If there is an arrow**: You automatically pick it up, increasing your arrow count.

**Controls (Keyboard)**

**To Move:**

* Use the **arrow keys** on your keyboard:
  + Up Arrow = move up
  + Down Arrow = move down
  + Left Arrow = move left
  + Right Arrow = move right

**To Shoot:**

* Hold the **SHIFT key** and press an arrow key in the direction you want to **fire an arrow**.

**Arrows and On-Screen Text**

At the top-left of the screen, you can always see:

* Your **current room number**
* Your **remaining number of arrows**
* Any nearby warnings like:
  + “You hear the squeaking of bats nearby”
  + “You feel a draft nearby”

You start with **1 arrow**, but you can find more in certain rooms.

**Goal of the Game**

Your goal is to **kill the Wumpus** by shooting it with an arrow.

If you:

* Walk into the Wumpus
* Fall into a pit
* Run out of arrows

...you lose.

If you shoot the Wumpus — you win the game.

## Artificial Intelligence Conversion Code

Explain what artificial intelligence engine your 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 made use of a machine learning engine referred to as CodeConvertAI to execute the Python programming into Java. The tool is focused on the translations of codes among programming languages without affecting the initial logic and structure of the code. It had been successful conversion and the CodeConvertAI did convert the Python code into the Java Syntax.

This allowed preserving the main part of functionality, but it was changed to the language that was more suited to the application of some of use cases like strong typing, or platform that underlies Java-based platforms. Software and hardware have their meaning in the context of speed software development.

The software side of technology has been incorporated with development acceleration tools like integrated development environment (IDE), code generation engine (CodeConvertAI), and version control system (e.g., GIT) that automates routine programs, assigns bugs early in development, and allows shared development. On the hardware aspect of the issue, the compile times are reduced, faster simulations, and the ability to multitask more efficiently because of the faster processors, larger amounts of memory, and faster storage device types. The hardware and software advances are designed to allow the builders to test and roll out applications in a more impulsive fashion and this is essential in an environment where the developer has to work with very tight deadlines and respond to the shifting requirements of the project in a rapid manner.

# Explanation of why Java is safer than Python or vice versa. Address in terms of Security, Compiling and Execution

Security

In the context of Hunt the Wumpus, security is not a major concern, since the game runs locally and doesn't interact with external systems or networks. Nonetheless, Java does provide more security by default in terms of stringent type safety, access controls and operating in the context of the Java Virtual Machine (JVM). These features make Java more secure by design, even if they aren't strictly necessary for a simple game.

Python, however, is more open to change and lenient. It is easier to program with, but this results in less safety checks being inbuilt, particularly in the form of type safety and memory handling. In a small-scale game such as Hunt the Wumpus however, the risks are low.

Compiling

Java is an objective-coupled language, which implies that a lot of errors can be identified prior to the execution of the program. This may result in more complex projects becoming safer. To Hunt the Wumpus, this assists in protecting against a crash in the logic of the game.

Python is of the interpreted type meaning that the code does not go through the compilation process. This enables faster tests and modifications, as is useful in a small game such as Hunt the Wumpus; the disadvantage is that bugs may not be found until run time, which may make debugging more difficult.

Execution

Java code is executed within JVM, which enables the creation of a controlled environment, which is more predictable and optimized. This can be a difference in the performance of bigger projects, but in simple turn-based games the difference is not very recognizable.

Python code is executed by the interpreter running directly, which is not a bad thing as far as Hunt the Wumpus is concerned, because the game does not require high performance or real-time reaction. The aspects of it being a bit slower and less strict do not pose a problem here.

Conclusion

In the case of a project as small as Hunt the Wumpus, Python is more suitable as we want to work and deploy as fast as possible, but the additional safety mechanisms available with Java costs more than it is worth. The game is neither sensitive to security nor does it need a rigid compilation, and it is performance-neutral, so Python has an advantage in terms of being easier to work with and conducting the necessary operations in the framework of this game.

## Storing data

Wumpus Game Using Data (OOP) Storing Data in the Wumpus Game (OOP Context)

In the Hunt the Wumpus project data like position of the player, cave structure, traps (bats, pits) and arrows are stored with the help of variables, lists, and maps. As used with Object-Oriented Programming (OOP), this data may be improved in its order; apart, it needs classes and objects.

For example:

Position and arrows could be stored in a class called a Player.

Room connections might be stored and rooms contents might be stored in a CaveRoom class.

It would be possible to have a Wumpus class that would have control over its behavior and position.

This whole process of managing and storing information using OOP helps improve the structure of the code, its reusability and scalability, particularly, as the game becomes more advanced. It helps to divide responsibilities and hold the data in a tight relationship to the objects it belongs to, enhancing maintainability and clarity.

## Encryption.

Wumpus Game (OOP Context) Encryption

Although the Hunt the Wumpus game may not experience the use of encryption currently, Object-Oriented Programming (OOP) allows one to add enhancement of encryption should the need arise, e.g. encryption of saved game or progress.

Implementing OOP, it might appear that a GameData or a SaveManager could be implemented to manage saving and loading of data. Hash algorithms (e.g. AES or Base64 encryption) may be an inbuilt feature of this class to ensure sensitive data.

The logic of encryption remains an encapsulated unit that is distinct and independent of game logic such as in adhering to the OOP concept of modularity, abstraction and encapsulation, hence simple enough to update or extend without necessarily touching the remaining part of the game.

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

The prototyping happens in Python not in Java, after all Python was designed to be simple, fast and flexible, and it fits the needs of swiftly constructing and testing concepts.

The syntax of Python is also very concise, meaning that the developer requires less code and devotes more time to functionality and not the boilerplate. This accelerates the development process enabling one to develop a working prototype of a program without having to waste time on set up and complex structure.

Java on the other hand is verbose and strictly typed and thus is slow to write and test even simple features. It improves the scalability of Java as an end-user final application, but it negatively affects the initial phases when the aim is to speculate, collect feedback, and develop concepts swiftly.

In a simplistic game such as Hunt the Wumpus, Python is the natural choice to quickly test various elements of game mechanics, structure and logic in a game. As soon as the prototype is working and the basic design is in place, it may be refactored to Java later (in the name of performance or deployment). However, Python is simply faster and easier to use at early stages of development thus makes a better prototyping language.

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

## Visual Code IDE

The Visual Studio Code (VS Code)

 VS Code has been employed as a lightweight and versatile editor when quickly making a change or performing a test in Python and Java. Extensions, and terminals that it supported enabled quick testing without a complete IDE. It was also helpful in editing auxiliary files such the README.

## PyCharm

PyCharm

 An application developed using PyCharm was Hunt the Wumpus version on Python. It also aided to structure the game logic, flag the syntax problems early, and was extremely generous when it came to supporting Pygame. Its pre-organized project exploration was beneficial in manipulation of images, sounds, and logic files.

## Artificial Intelligence Converter

CodeConvert AI

 CodeConvert AI was applied to help translate Python into Java code, in particular logic structures, loops, and functions behavior. It reduced time spent during the early period of language translation, and it substantially contributed to keeping the essence of the game mechanics similar in both versions.

## Java: IntelliJ

IntelliJ IDE

 Testing and writing of the Java version of the game was done in IntelliJ. It offers code intelligence, error verification and simple GUI support such as Swing elements. It had an in-built debugger as well as project structure tool through which it was simpler and faster to control and polish the Java code.

## Code is commented and following industry standard practices

The *Hunt the Wumpus* code is well-commented throughout, with clear explanations for functions, variables, and logic blocks. These comments help anyone reading the code to quickly understand the purpose of each section, how key functions operate, and the flow of the game. This level of documentation is essential for maintainability, debugging, and future development.

Additionally, the code follows industry standard practices such as:

* Meaningful variable and function names that describe their role clearly (e.g., player\_pos, check\_room(), shoot\_arrow()).
* Modular design by organizing functionality into separate functions or methods to keep the code organized and reusable.
* Use of constants for fixed values like screen size, colors, and directions, improving readability and ease of changes.
* Consistent indentation and formatting which enhances readability and aligns with language-specific style guides.
* Error handling and edge case consideration, such as checking for invalid moves or ensuring the Wumpus does not spawn in the same room as the player.
* Separation of concerns by handling user input, game state updates, and rendering in distinct code areas or methods.

Together, these practices make the code professional, easy to follow, and adaptable for further enhancements or collaboration.

## Justification of Git and GitHub and their difference

During the development of the game Hunt the Wumpus and its conversion to Java, the utilization of the Git and GitHub became the effective tools to handle the project professionally and efficiently. The Difference between Git and GitHub, Git is a version system that is executed in your computer. It lets you save your work in stages (called commits), go back to earlier versions, and experiment safely — for example, testing a new feature like mobile Wumpus behavior without breaking the working game. GitHub is an internet-based portal where you can put your Git repository. It also enables you to save, share, or collaborate your project with other people, or even have access to it on other devices.

**The Reason They were Useful in Hunt the Wumpus:**

**Version Tracking**

* Git also helped me to keep tabs on all the changes when new features such as sound effects were incorporated, GUI was drawn and when the player was moved by the bats. In case a new feature broke up the game, then I could just go back to a previous working version.

**Python to Java conversion**

* Somewhere along the process of converting the language, Git enabled me to work through a clean separation in breaking out the Python and Java codebases, thereby enabling the ability to commit and test an individual portion of the Java implementation without affecting the original code.

**Testing and Experimentation**

* The things I tested with branches in Git were how to better render the GUI or new hazards, without touching the main version. This assisted in evading bugs and modifying it with assurance.

**Posting at GitHub**

 GitHub enabled me to launch the project on the internet so that it is possible to collaborate or present it. It is easy to access the code and history of the projects by people like teachers, classmates, or even prospective employers.

## Frequency of committing Code

The frequent commitments in the Hunt the Wumpus project were important because with each implementation of the Object-Oriented Programming (OOP) concepts, the version of Java was implemented.

Since new classes like Player, Wumpus, or GamePanel come into play, small frequent commits enabled one to keep an eye on the progress, test each object separately, as well as roll back in case it went wrong with ease. Such commit approach aids in OOP development because it allows dealing with changes in encapsulated objects to remain controllable and traceable, leading to clean and dependable codebase.

# 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

dodecahedron 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,

specifically 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 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

# Hunt the Wumpus (Python Game)

This is a graphical version of the classic \*\*Hunt the Wumpus\*\* game, built using \*\*Python and Pygame\*\*.

You are trapped in a 20-room cave filled with hidden dangers: a fearsome Wumpus, giant bats, and bottomless pits. Your objective is to hunt and kill the Wumpus using arrows — but be careful! One wrong move, and it could be the end.

---

## Features

- Graphical cave layout using Pygame

- Randomly placed:

- Wumpus (the monster)

- Bats (which move you randomly)

- Pits (which kill you instantly)

- Extra arrows to collect

- On-screen text hints:

- “You hear the squeaking of bats nearby” → bats are close

- “You feel a draft nearby” → pit is close

- Red screen glow → Wumpus is nearby

---

## How to Play

- Use the \*\*arrow keys\*\* to move between rooms.

- Use \*\*Shift + arrow key\*\* to shoot an arrow in that direction.

- Avoid falling into pits or landing in the Wumpus's room.

- If you run into bats, they’ll pick you up and drop you in a random room.

- If you run out of arrows or walk into the Wumpus, the game ends.

- If you hit the Wumpus with an arrow, you win.

---

## Controls

- \*\*Arrow keys\*\* — Move in a direction

- \*\*Shift + Arrow key\*\* — Shoot an arrow

- \*\*Escape\*\* — Exit the game

---

## Objective

Survive the cave and kill the Wumpus before it kills you.

# Appendix 2 Java Code

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

# Hunt the Wumpus (Java Game)

This is a Java-based graphical remake of the classic game \*\*Hunt the Wumpus\*\*. The game uses \*\*Swing\*\* and \*\*AWT\*\* for graphics and interface, allowing players to explore a cave, avoid hidden threats, and hunt the legendary Wumpus.

---

## Features

- Graphical interface with custom images and sounds

- Randomized cave with:

- Bats that move the player to a new location

- Bottomless pits that instantly end the game

- Arrows to collect and shoot

- Wumpus that moves after missed shots

- Sound effects for key events (victory, pitfall, arrow pickup, etc.)

- Visual feedback:

- Bloodstains = Wumpus nearby

- Draft = pit nearby

- Squeaks = bats nearby

- Restart option in both the menu and with the `R` key

---

## How to Play

- You start in one of 20 interconnected rooms.

- Your mission: \*\*Kill the Wumpus\*\* before it kills you.

- You begin with one arrow. More can be found in some rooms.

- Be cautious:

- \*\*If you enter a room with the Wumpus\*\*, it eats you.

- \*\*If you fall into a pit\*\*, you die.

- \*\*If you run into bats\*\*, they move you (and themselves) to a random room.

- Use clues to survive:

- “You hear bats” = bats nearby

- “You feel a draft” = pit nearby

- “You see bloodstains” = Wumpus nearby

---

## Controls

- \*\*Arrow Keys\*\* — Move to an adjacent room

- \*\*Shift + Arrow Key\*\* — Shoot an arrow in that direction

- \*\*R\*\* — Restart the game after game over

- \*\*Game > Restart\*\* — Also restarts the game via the menu

---

## Objective

Survive the cave and defeat the Wumpus with your arrows. Make smart moves based on visual and text clues — and good luck!

# Reflection

Besides the objective of replicating the Hunt the Wumpus game in Java as it was originally coded in Python, the exercise offered me a worthwhile chance to attain a better grasp of the concept of Object-Oriented Programming (OOP). Although the Python version was working properly and was written adhering to procedural programming and global variables, due to the shift to Java I had to think more structure-wise with such concepts as objects, classes and encapsulation.

Among the most valuable things one can learn was the matter of layering the logic into classes. Since java is a strictly object-oriented language, rendering the game had to be done using a JPanel subclass, and integrating the main game loop with Java Swing Event handling system was required. This transition was an eye-opener to show me that OOP enables more definite segregation of duties and responsibilities eg: easier to maintain the game state, display and system input states as independent and synchronized with each other.

And also, I understood that encapsulation and data hiding help to make code easier to maintain. Instead of relying on global variables as in Python, Java encouraged me to keep game data like the player's position, arrow count, and Wumpus state within the class. This gave more clarity, and she minimized undesired side effects.

Working with event-driven programming in Java (through KeyListener) contrasted with Python's more loop-based approach (using Pygame). I also got to know how to handle asynchronous input and repaint cycles by calling methods such as repaint () and how to maintain the game responsive by using listener callbacks.

Finally, I learned more about the constructors, methods, and interactions of objects as the way to simplify behavior through the complex behavior (such as moving bats or checking hazards) so that it makes it easier to process can be broken down into simpler part, which is in line with the main OOP principles, such as modularity and reusability.

All in all, the language-specific syntaxes taught by the conversion of this game were not the only ones I learned as OOP principles and helped me to better understand how to design software, particularly when speaking of a graphical, user-interactive application such as this one.