**Software Engineering**

**Year 11 , 2025**

**Assessment Task 2**

**Object-Oriented Programming Assignment:**

**“Hunting Wumpus”**

**By: Kaleb Vong 11SENG1P**

Contents

[Storyboard (What is the Game about?) 3](#_Toc202193866)

[Gantt Chart 4](#_Toc202193867)

[Resource Allocation, Budget & Manpower 5](#_Toc202193868)

[Justification of Technology 6](#_Toc202193869)

[Design 7](#_Toc202193870)

[UML Notation 8](#_Toc202193871)

[Class Diagram 9](#_Toc202193872)

[Flowchart 10](#_Toc202193873)

[Pseudocode 11](#_Toc202193874)

[Sequence Diagram 13](#_Toc202193875)

[Context Diagram 14](#_Toc202193876)

[Use Case Diagram 14](#_Toc202193877)

[Graphical User Interface GUI 15](#_Toc202193878)

[Artificial Intelligence Conversion Code 16](#_Toc202193879)

[Explanation of why Java is safer than Python or vice versa 17](#_Toc202193880)

[Security 17](#_Toc202193881)

[Compiling and Execution 17](#_Toc202193882)

[Storing data 18](#_Toc202193883)

[Encryption. 19](#_Toc202193884)

[Why prototyping might be done in Python rather than Java. 20](#_Toc202193885)

[What Tools were used in the development of this Project and their justification 21](#_Toc202193886)

[Visual Studio Code IDE 21](#_Toc202193887)

[Draw.io Drawing Tool 22](#_Toc202193888)

[Artificial Intelligence 23](#_Toc202193889)

[Java 24](#_Toc202193890)

[Code is commented and following industry standard practices 25](#_Toc202193891)

[Justification of Git and GitHub and their difference 26](#_Toc202193892)

[Frequency of committing Code 27](#_Toc202193893)

[Appendix 1 Python Code 28](#_Toc202193894)

[Readme File for Python 36](#_Toc202193895)

[Appendix 2 Java Code 37](#_Toc202193896)

[Readme File for Java 44](#_Toc202193897)

[Reflection 45](#_Toc202193898)

# Storyboard (What is the Game about?)

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

Hunt The Wumpus is an enthralling, single-player roleplay game (RPG) featuring aspects of adventure and mild-horror. The gameplay consists of the player exploring a maze-like cave, armed with a bow and only one arrow. They must use this arrow to shoot and kill the Wumpus, which lurks in one of the chambers of the cave. This cave doesn’t just house a Wumpus however, bottomless pits you can fall into, player-transporting bats and rooms stained with blood all populate this cave of horrors.

## Gantt Chart

*An accurate track record of what you did for the project. This can be recorded from Git or GitHub.*

## Resource Allocation, Budget & Manpower

*You are a software engineer charging $60 per hour. Get the time spent from GitHub and multiply by $60 per hour.*

***Resource Allocation***

Throughout the development of this digital project, numerous online resources were allocated and used to increase rates of development, avoid internal complications with the project such as software bugs, and to make the task of programming easier. Resources such as first coding in the Python programming language were allocated as this language is designed around a philosophy of human readability and being amateur-friendly. Then, artificial intelligence had been allocated to convert this code into Java code to reduce the workload of programming.

***Budget***

This project had taken me approximately 20 hours of time spent planning, developing, playtesting and maintenance work according to GitHub, the main online code repository I had used to ensure version control. This time was over the course of 3 weeks, with an average of one hour developing this project each day. As such, if I were to get paid $60 per hour, I would end up with $1200 dollars in compensation for this work.

***Manpower***

The manpower of this project only consisted of one sole developer. As such, all roles not supplemented by other online sources (such as the role of **artist**) were solely partaken by this developer. Examples of the roles needed to be taken to create this project were **UI Designer**, **Main Coder** and **Playtester**. However, the use of Artificial Intelligence (AI) had been utilized for increased rates of development of the project and to supplement the need of a multiple-person development team.

## 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.*

Python is a highly versatile and easily learnable programming language designed around a philosophy of human readability and flexibility. Many programmers have developed third-party library extensions for this programming language for other developers to use and thus increasing the range of opportunities for programs coded in Python to take. Meanwhile, other programming languages such as Java are more behind with this technology, usually possessing sophisticated structures with limited human readability.

The IDE (integrated developed environment) utilised during the creation of this project was Visual Studio Code. I had used this IDE as I am the most familiar with the layout of program and have the most experience with it. Advantages of this IDE programs over other programs are its lightweight nature and extensive community support and listening to feedback of developers.

# Design

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

## UML Notation

*Why do we use UML Notation?*

UML (Unified Modelling Language) notation is used to visually represent systems and software, aiding in design, development, and communication. It simplifies complex systems and helps visualize code. Thus, it improves collaboration amongst multiple developers when working on the same project. UML diagrams essentially act as a common language, ensuring everyone understands the system's design and progress.

## Class Diagram

A diagram of a computer

AI-generated content may be incorrect.

## A diagram of a flowchartFlowchart

## Pseudocode

INITIALIZE constants:

DIRECTIONS = [UP, DOWN, LEFT, RIGHT]

COLORS = {BROWN, BLACK, RED}

GAME SETTINGS: NUM\_BATS = 3, NUM\_PITS = 3, NUM\_ARROWS = 1

WUMPUS\_MOVES = FALSE, MOVE\_CHANCE = 50%

WINDOW\_SIZE = 800x600

CAVE\_LAYOUT = Map of 20 rooms, each with 4 connected directions

DECLARE game state variables:

playerPosition, wumpusPosition

numberOfArrows

batsList, pitsList, arrowsList

messageToPlayer

randomGenerator

LOAD player sprites (up, down, left, right)

IF loading fails THEN print error

SET window size and background

SET game to listen to keyboard

CALL resetGame()

FUNCTION resetGame():

CLEAR bats, pits, arrows

SET numberOfArrows = NUM\_ARROWS

PLACE player in a random room

PLACE Wumpus in a different random room

FOR i from 1 to NUM\_BATS:

PLACE bat in a random room (not player, wumpus, or other bats/pits)

FOR i from 1 to NUM\_PITS:

PLACE pit in a random room (same exclusions)

FOR i from 1 to NUM\_ARROWS:

PLACE arrow in a random room (same exclusions)

DISPLAY welcome message

FUNCTION checkRoom():

IF player is in same room as Wumpus:

CALL gameOver("Wumpus ate you!")

IF player falls in a pit:

CALL gameOver("You fell into a pit!")

IF player encounters bats:

DISPLAY message

WAIT 2 seconds

REMOVE bat from room

MOVE bat to a new valid room

TELEPORT player to a new valid room

DISPLAY updated message

IF player finds arrow:

DISPLAY message

WAIT 1.5 seconds

REMOVE arrow from room

INCREMENT arrow count

FUNCTION gameOver(message):

DISPLAY message in popup

TERMINATE game

FUNCTION moveWumpus():

IF Wumpus is not mobile THEN return

IF random number > WUMPUS\_MOVE\_CHANCE THEN return

FOR each connected room to Wumpus:

IF room is valid AND not player, bat, or pit:

MOVE Wumpus to this room

BREAK

FUNCTION shootArrow(direction):

IF no arrows left THEN

message = "No arrows left!"

RETURN

DECREMENT arrow count

SET targetRoom = room in direction from player

IF targetRoom == wumpus:

CALL gameOver("You killed the Wumpus!")

ELSE:

message = "Your arrow missed..."

RELOCATE Wumpus

IF no arrows remaining:

CALL gameOver("You ran out of arrows and died!")

FUNCTION checkNeighborRooms(position, hazardList):

FOR each room connected to position:

IF room is in hazardList:

RETURN true

RETURN false

FUNCTION paintComponent(graphics):

DRAW black background

DRAW brown circular room at center

DRAW exits (brown rectangles) based on available directions

IF nearby Wumpus:

DRAW red warning circle

IF nearby bats:

DISPLAY "You hear bats squeaking"

IF nearby pits:

DISPLAY "You feel a draft"

DRAW player image pointing based on direction

DISPLAY HUD: current room, arrows remaining, message

FUNCTION movePlayer(direction):

GET destination room in that direction

IF room is invalid:

message = "You can't move that way!"

ELSE:

UPDATE player position and direction

message = "You moved to room X"

CALL checkRoom()

CALL moveWumpus()

REFRESH screen

FUNCTION keyPressed(event):

DETECT if Shift key is held

READ arrow key pressed

IF Shift is not held:

CALL movePlayer(direction)

ELSE:

CALL shootArrow(direction)

REFRESH screen

FUNCTION keyReleased(event): DO NOTHING

FUNCTION keyTyped(event): DO NOTHING

MAIN FUNCTION:

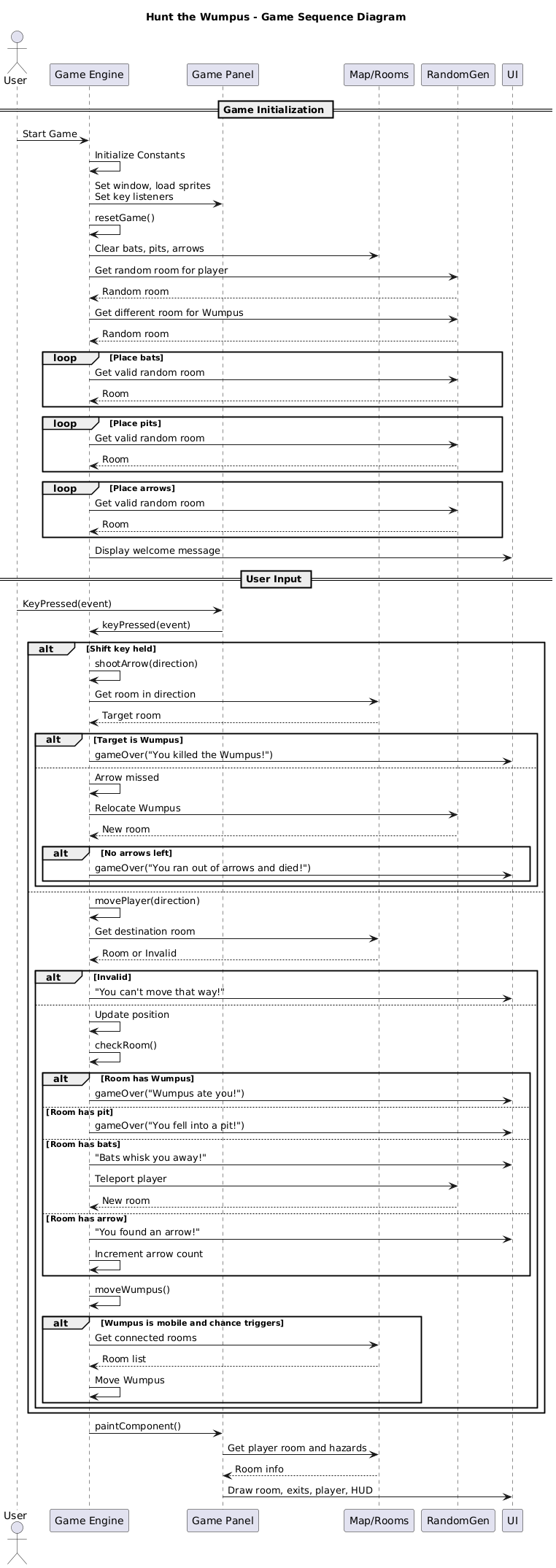
CREATE window titled "Hunt the Wumpus"

INITIALIZE game panel

ATTACH panel to window

SET window location and visibility

## Sequence Diagram



## Context Diagram

A diagram of a game

AI-generated content may be incorrect.

## Use Case Diagram

A diagram of a game

AI-generated content may be incorrect.

## Graphical User Interface GUI

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

## 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.*

The model of artificial intelligence I had used to convert my Python Code into Java Code was **ChatGPT**. Despite being mostly renowned for being an intelligent chatbot, it can also be used to help with programming techniques such as code optimisation and thus can modify a program’s code. As such, ChatGPT had been used to modifying the entire Python Hunt the Wumpus program and successfully turned it into a program coded in Java – despite there being major differences between the two languages. This ability with artificial intelligence to convert code will absolutely increase the speed of software development and will revolutionise the software industry with practices of rapid software development – making the field less intensive and easier for beginners to get into.

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

Address in terms of:

# Security

### Compiling and Execution

The Java coding language is safer than Python because of the difference between their **typing streams** when executed and how programs written in these languages **check compiling times**.

Java utilises a static typing stream with variables being checked early at compiling time – when the source code of a program is being translated into machine code via a compiler. As such, type-related errors within the code can be caught early before the code is even executed. Meanwhile, Python uses a dynamic typing stream unlike Java – which cause type-related errors to show up only when the code is running. Thus, potential runtime crashes can occur with Python if these errors aren’t carefully handled before execution.

When Java code in compiled by a compiled, it is converted into bytecode and thus checked for syntax, type safety and other issues. Meanwhile, Python code is merely interpreted by the machine, leading to errors only being caught when the specific line the error is in is executed by the program.

## Storing data

The Java programming language has a massive upside to Python in terms of data storage within a program as it heavily relies on the usage of **access modifiers** and **encapsulation**.

As Java is a fully class-based programming language, it fully relies on the object-oriented paradigm of coding and the flexibility of classes. Java’s static typing stream ensures data types are declared explicitly and thus reduces risks of type-related errors or unintended code manipulation.

Meanwhile, Java’s heavy usage of access modifiers and encapsulation of data complements its static typing stream in reducing unintended code manipulation as it enforces levels of authorisation to access of data stored within classes. Hence, depending on the access modifier a class is instantiated with, data stored (encapsulated) within this class ranges from being completely visible and manipulatable by other classes to being completely hidden.

While Python does have the ability to encapsulate data within classes, it does not have access modifiers to set the level of authority needed to access this data, thus making Java ultimately superior in terms of data storage within a program.

## Encryption.

While Python is built around a design philosophy of human readability and being a beginner friendly programming language, Java sacrifices these design aspects for security and cyber-safety.

As Java has been used for numerous years within many enterprise industries, the programming language has had numerous **security tools and frameworks** to be built upon it that hinder cybernetic attacks such as data tampering and having unauthorised access to datasets. Examples of security tools built with Java include The Java Cryptography Architecture (JCA) and Java Authentication and Authorization Service (JAAS). While Python also has security tools of their own, they can all vary in quality as Python has not been utilised in enterprise environments for as long as Java.

Java also includes **immutability support** through “final” variables, immutable classes and third-party library extensions such as Guava and Lombok. Hence, these features of Java all prevent unintended side effects and ensures thread safety during compile time.

Finally, Java addresses the need of **concurrency safety** through its possession of native multithreading support. It features powerful concurrency utilities such as java.util.concurrent and ReentrantLock that directly lock to OS threads, providing true parallelism and finely tuned control.

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

While Java may be a far more secure language than Python, prototyping coding projects may be done in Python instead of Java as the need for security is redundant in this phase of development.

Python’s concise and easily readable syntax structure means less code can be written by developers to perform more. As such, Python directly supports the rapid development of software and thus makes it the ideal programming language for prototyping ideas or proof of concepts.

Python also includes a vast range of third-party libraries to increase its range of use cases. Libraries such as Flask and Django all help with web development, meanwhile other libraries such as NumPy, pandas and scikit-learn all allow for data science programs to be all written with Python. This vast variety of use cases further attract developers to prototype ideas or proof-of-concepts with Python, then switch it to Java when security becomes a concern later down the development cycle.

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

## Visual Studio Code IDE

The Integrated Development Environment I had used to develop the Hunt the Wumpus game was Visual Studio Code as it fastens development of projects via its extensive capabilities and characteristics.

Visual Studio Code is an incredibly lightweight application – it launches extremely fast and doesn’t take up too much CPU, Memory or Disk Space when running on your system.

Furthermore, its built-in support for syntax highlighting for almost all coding languages, code and workspace navigation, refactoring tools and interactive debugging all provide an organised yet productive environment to write code in, further justifying the usage of this program during the creation process of this game.

## Draw.io Drawing Tool

To create all the diagrams and visual representations needed to aid with the development of the Hunt the Wumpus program, I had used the free, online website Draw.io due to its ability to create a wide range of diagrams and visual representations.

Draw.io features many base templates for diagrams such as class diagrams and flowcharts. As such, I utilised this website to rapidly create and aid the creation of my own code and procedure visualisations.

Furthermore, all Draw.io programs have the option to be exported as a PNG image file and be saved to your personal Google Drive. Thus, the process of transporting my diagrams to this document had been done extremely quickly without the need of any other third-party utilities such as Snipping Tool.

## Artificial Intelligence

The Artificial Intelligence program I had used to convert my code from the Python programming language to Java was ChatGPT.

Despite being an online chatbot and not directly designed for this purpose, I had utilised the program after attempting to use other programs such as CodeConvert.AI and Microsoft’s Copilot. However, both programs had their own retrospective issues that ChatGPT did not have.

CodeConvert.AI had a 4000-character limit for programs, and this limit could only be bypassed with a Monthly or Yearly Subscription. Meanwhile, Copilot, while being completely free, also had a character limit, being 10240-characters only.

These character limits would have caused my code to be structured completely incorrectly, as both Python and Java are read by a machine procedurally from top to bottom – missing key elements from previous lines would affect all lines that come next. However, ChatGPT did not have a character limit and as such I had used it for this project to convert my code without running into any issues.

## Java

The Java programming language had been used as it comprises of a robust and secure design. Within Java, code is securely encapsulated and robustly stored within classes – and these classes are defined with access modifiers that determine the level of authority needed to access this class. As such, Java had been used to ensure no unwanted and unintended data tampering and malware could be present or done within my Hunt the Wumpus program.

## Code is commented and following industry standard practices

Both the Python and Java code of the Hunt the Wumpus program had been commented to follow industry standards. Despite the program only being developed by one sole developer, the code had been frequently commented justifying what the functions of lines of code were without the need to interpret the code into pseudocode. As such, comments were used to dramatically increase the human readability of both the Java and Python programs.

## Justification of Git and GitHub and their difference

Both Git and GitHub were utilised in my Hunt the Wumpus game project as they both offered necessary features for this project. I had used Git to ensure version control during the coding phase of the game – I had numerous instances where code had completely broken and needed to roll back to a previous version of my code.

Meanwhile, I had also used GitHub as a remote, online repository to store these distinct versions of my code, allowing for me to simply redownload older versions of the same file.

The difference between Git and GitHub is that Git acts as the tool that enables version control, tracking whenever you make changes to your code and allowing for collaboration management through branches and merging. Meanwhile, GitHub is the platform that stores these features of Git, allowing for developers to upload versions of their code to an online repository and making it open-source, allowing for others to create pull-requests and modifying this code to their own liking.

## Frequency of committing Code

During the creation of my Hunt the Wumpus program, the frequency of committing code to the online GitHub repository for the game was quite frequent, with new versions of the game being uploaded every couple of days. I did this to ensure that there were not too many files uploaded to the GitHub that only had few minor changes – but rather files only be on GitHub that had substantial changes between versions.

# Appendix 1 Python Code

import pygame

import random

import time

import sys

### DEFINE FUNCTIONS

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 circles in brown

circle\_radius = int ((SCREEN\_WIDTH//3)\*.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//2-100

pygame.draw.rect(screen, BROWN, ( (left,top), (SCREEN\_WIDTH//3.6,200)), 0)

if exits[RIGHT] > 0:

# draw right exit

left = SCREEN\_WIDTH-(SCREEN\_WIDTH//3.6)

top = SCREEN\_HEIGHT//2-100

pygame.draw.rect(screen, BROWN, ((left,top), (SCREEN\_WIDTH//3.6,200)), 0)

if exits[UP] > 0:

# draw top exit

left = SCREEN\_WIDTH//2-100

top = 0

pygame.draw.rect(screen, BROWN, ((left,top), (200,SCREEN\_HEIGHT//3.6)), 0)

if exits[DOWN] > 0 :

# draw bottom exit

left = SCREEN\_WIDTH//2-100

top = SCREEN\_HEIGHT-(SCREEN\_WIDTH//10)

pygame.draw.rect(screen, BROWN, ((left,top), (200,SCREEN\_HEIGHT//3.6)), 0)

# check if hazards are nearby

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 blood circle if the Wumpus is nearby

if wumpus\_near == True:

circle\_radius = int ((SCREEN\_WIDTH//2)\*.4)

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

# draw pit in black if it is present

if player\_pos in pits\_list:

circle\_radius = int ((SCREEN\_WIDTH//2)\*.4)

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

# load player images

screen.blit(

player\_img,

(SCREEN\_WIDTH//2 - player\_img.get\_width()//2,

SCREEN\_HEIGHT//2 - player\_img.get\_height()//2)

)

# load bat image

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))

# load wumpus image

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:

time.sleep(1.0)

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, player\_img

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]

player\_img = player\_left\_img

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]

player\_img = player\_right\_img

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]

player\_img = player\_up\_img

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]

player\_img = player\_down\_img

move\_wumpus()

def print\_instructions():

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.

'''

)

### DEFINE GLOBALS AND CONSTANTS

#load our three images

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

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

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

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

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

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

player\_down\_img = pygame.image.load('images/player\_down.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 = 1

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

# colour 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 = []

### INITIALISE GAME

print\_instructions()

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

pygame.init()

# Our screen width and height

display\_info = pygame.display.Info()

SCREEN\_WIDTH = display\_info.current\_w

SCREEN\_HEIGHT = display\_info.current\_h

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

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

# load our images

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

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

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

# load all player images

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

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

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

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

# load default player image

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

# load 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

1. Open Visual Studio Code or any other Integrated Development Environment that supports running Python Code
2. Open the **Folder** which includes the **main.py** and **images** files **via the IDE**
3. Navigate to the **main.py** file within the IDE
4. Run the file

# Appendix 2 Java Code

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.util.\*;

import java.util.List;

public class HuntTheWumpus extends JPanel implements KeyListener {

// Constants for directions

private static final int UP = 0;

private static final int DOWN = 1;

private static final int LEFT = 2;

private static final int RIGHT = 3;

// Colours

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

private static final Color BLACK = Color.BLACK;

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

// Game settings

private static final int NUM\_BATS = 3;

private static final int NUM\_PITS = 3;

private static final int NUM\_ARROWS = 1;

private static final boolean MOBILE\_WUMPUS = false;

private static final int WUMPUS\_MOVE\_CHANCE = 50; // %

// Cave layout: Map<Room, List of connected rooms [UP, DOWN, LEFT, RIGHT]>

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

static {

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

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

cave.put(3, new int[]{0,12,4,2});

cave.put(4, new int[]{0,14,5,3});

cave.put(5, new int[]{0,6,1,4});

cave.put(6, new int[]{5,0,7,15});

cave.put(7, new int[]{0,17,8,6});

cave.put(8, new int[]{1,0,9,7});

cave.put(9, new int[]{0,18,10,8});

cave.put(10,new int[]{2,0,11,9});

cave.put(11,new int[]{0,19,12,10});

cave.put(12,new int[]{3,0,13,11});

cave.put(13,new int[]{0,20,14,12});

cave.put(14,new int[]{4,0,15,13});

cave.put(15,new int[]{0,16,6,14});

cave.put(16,new int[]{15,0,17,20});

cave.put(17,new int[]{7,0,18,16});

cave.put(18,new int[]{9,0,19,17});

cave.put(19,new int[]{11,0,20,18});

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

}

// Game state variables

private int playerPos;

private int wumpusPos;

private int numArrows;

private final List<Integer> batsList = new ArrayList<>();

private final List<Integer> pitsList = new ArrayList<>();

private final List<Integer> arrowsList = new ArrayList<>();

private Random random = new Random();

// Player sprites

private Image playerUpImage;

private Image playerDownImage;

private Image playerLeftImage;

private Image playerRightImage;

{

try {

playerUpImage = new ImageIcon("images/player\_up.png").getImage();

playerDownImage = new ImageIcon("images/player\_down.png").getImage();

playerLeftImage = new ImageIcon("images/player\_left.png").getImage();

playerRightImage = new ImageIcon("images/player.png").getImage();

}

catch (Exception e) {

System.err.println("Failed to load player images.");

}

}

// Window size

private int width = 800;

private int height = 600;

// Game messages

private String message = "";

private int playerDirection;

// Main constructor

public HuntTheWumpus() {

setPreferredSize(new Dimension(width, height));

setBackground(BLACK);

setFocusable(true);

addKeyListener(this);

resetGame();

}

// Reset game state

private void resetGame() {

batsList.clear();

pitsList.clear();

arrowsList.clear();

numArrows = NUM\_ARROWS;

// Place player

playerPos = random.nextInt(20) + 1;

// Place wumpus

placeWumpus();

// Place bats

for (int i=0; i<NUM\_BATS; i++) placeBat();

// Place pits

for (int i=0; i<NUM\_PITS; i++) placePit();

// Place arrows

for (int i=0; i<NUM\_ARROWS; i++) placeArrow();

message = "Welcome to Hunt the Wumpus! Use arrow keys to move. Shift + arrow to shoot.";

}

// Place wumpus in a random room not the player's

private void placeWumpus() {

wumpusPos = playerPos;

while (wumpusPos == playerPos) {

wumpusPos = random.nextInt(20) + 1;

}

}

private void placeBat() {

int pos;

do {

pos = random.nextInt(20) + 1;

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

batsList.add(pos);

}

private void placePit() {

int pos;

do {

pos = random.nextInt(20) + 1;

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

pitsList.add(pos);

}

private void placeArrow() {

int pos;

do {

pos = random.nextInt(20) + 1;

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

arrowsList.add(pos);

}

// Check hazards in the room player moved to

private void checkRoom() {

if (playerPos == wumpusPos) {

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

}

if (pitsList.contains(playerPos)) {

gameOver("You fell into a bottomless pit!!");

}

if (batsList.contains(playerPos)) {

message = "Bats pick you up and drop you elsewhere!";

repaint();

try { Thread.sleep(2000); } catch (InterruptedException ignored) {}

// Move the bats (remove current bat)

batsList.remove((Integer)playerPos);

int newBatPos;

do {

newBatPos = random.nextInt(20) + 1;

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

batsList.add(newBatPos);

// Move the player

int newPlayerPos;

do {

newPlayerPos = random.nextInt(20) + 1;

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

playerPos = newPlayerPos;

message = "You are now in room " + playerPos;

}

if (arrowsList.contains(playerPos)) {

message = "You have found an arrow!";

repaint();

try { Thread.sleep(1500); } catch (InterruptedException ignored) {}

arrowsList.remove((Integer)playerPos);

numArrows++;

}

}

// Game over message and exit

private void gameOver(String msg) {

JOptionPane.showMessageDialog(this, msg, "Game Over", JOptionPane.INFORMATION\_MESSAGE);

System.exit(0);

}

// Move the wumpus randomly if allowed

private void moveWumpus() {

if (!MOBILE\_WUMPUS) return;

if (random.nextInt(100) > WUMPUS\_MOVE\_CHANCE) return;

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

for (int newRoom : exits) {

if (newRoom == 0 || newRoom == playerPos) continue;

if (batsList.contains(newRoom)) continue;

if (pitsList.contains(newRoom)) continue;

wumpusPos = newRoom;

break;

}

}

// Shoot an arrow in given direction

private void shootArrow(int direction) {

if (numArrows == 0) {

message = "No arrows left!";

return;

}

numArrows--;

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

int targetRoom = exits[direction];

if (targetRoom == wumpusPos) {

gameOver("Your aim was true and you killed the Wumpus!");

} else {

message = "Your arrow missed...";

placeWumpus(); // Move wumpus

}

if (numArrows == 0) {

gameOver("You ran out of arrows and died!");

}

}

// Check neighbors for hazards (used for warning messages)

private boolean checkNeighborRooms(int pos, List<Integer> list) {

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

for (int room : exits) {

if (room > 0 && list.contains(room)) return true;

}

return false;

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

// Background

g.setColor(BLACK);

g.fillRect(0, 0, getWidth(), getHeight());

// Draw room circle (brown)

int radius = Math.min(getWidth(), getHeight()) / 3;

g.setColor(BROWN);

g.fillOval(getWidth()/2 - radius, getHeight()/2 - radius, radius\*2, radius\*2);

// Draw exits

int exitWidth = 80;

int exitLength = getWidth() / 2;

int centerX = getWidth() / 2;

int centerY = getHeight() / 2;

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

g.setColor(BROWN);

if (exits[LEFT] > 0) {

g.fillRect(centerX - radius - exitLength, centerY - exitWidth / 2, exitLength, exitWidth);

}

if (exits[RIGHT] > 0) {

g.fillRect(centerX + radius, centerY - exitWidth / 2, exitLength, exitWidth);

}

if (exits[UP] > 0) {

g.fillRect(centerX - exitWidth / 2, centerY - radius - exitLength, exitWidth, exitLength);

}

if (exits[DOWN] > 0) {

g.fillRect(centerX - exitWidth / 2, centerY + radius, exitWidth, exitLength);

}

// Draw warnings for hazards nearby

if (checkNeighborRooms(playerPos, Collections.singletonList(wumpusPos))) {

g.setColor(RED);

g.fillOval(getWidth()/2 - radius/2, getHeight()/2 - radius/2, radius, radius);

}

if (checkNeighborRooms(playerPos, batsList)) {

g.setColor(Color.WHITE);

g.drawString("You hear the squeaking of bats nearby", 10, 20);

}

if (checkNeighborRooms(playerPos, pitsList)) {

g.setColor(Color.WHITE);

g.drawString("You feel a draft from a nearby pit", 10, 40);

}

// Draw player pointing direction depending on key input

Image imgToDraw = null;

switch (playerDirection) {

case UP:

imgToDraw = playerUpImage;

break;

case DOWN:

imgToDraw = playerDownImage;

break;

case LEFT:

imgToDraw = playerLeftImage;

break;

case RIGHT:

imgToDraw = playerRightImage;

break;

}

if (imgToDraw != null) {

// Draw image centered at centerX, centerY

int imgWidth = imgToDraw.getWidth(null);

int imgHeight = imgToDraw.getHeight(null);

g.drawImage(imgToDraw, centerX - imgWidth/2, centerY - imgHeight/2, null);

}

// Draw HUD info

g.setColor(Color.WHITE);

g.drawString("Room: " + playerPos, 10, getHeight() - 60);

g.drawString("Arrows: " + numArrows, 10, getHeight() - 40);

g.drawString(message, 10, getHeight() - 20);

}

// Move player if possible

private void movePlayer(int direction) {

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

int nextRoom = exits[direction];

if (nextRoom == 0) {

message = "You can't move that way!";

} else {

playerPos = nextRoom;

playerDirection = direction;

message = "You moved to room " + playerPos;

checkRoom();

moveWumpus();

}

repaint();

}

// Handle key presses

@Override

public void keyPressed(KeyEvent e) {

boolean shift = e.isShiftDown();

int key = e.getKeyCode();

if (!shift) {

switch (key) {

case KeyEvent.VK\_UP: movePlayer(UP); break;

case KeyEvent.VK\_DOWN: movePlayer(DOWN); break;

case KeyEvent.VK\_LEFT: movePlayer(LEFT); break;

case KeyEvent.VK\_RIGHT: movePlayer(RIGHT); break;

}

} else {

switch (key) {

case KeyEvent.VK\_UP: shootArrow(UP); break;

case KeyEvent.VK\_DOWN: shootArrow(DOWN); break;

case KeyEvent.VK\_LEFT: shootArrow(LEFT); break;

case KeyEvent.VK\_RIGHT: shootArrow(RIGHT); break;

}

repaint();

}

}

@Override

public void keyReleased(KeyEvent e) {}

@Override

public void keyTyped(KeyEvent e) {}

// Main method to start the game

public static void main(String[] args) {

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

HuntTheWumpus game = new HuntTheWumpus();

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

frame.setContentPane(game);

frame.pack();

frame.setLocationRelativeTo(null);

frame.setVisible(true);

}

}

## Readme File for Java

1. Open the **Folder** that contains the **HuntTheWumpus.exe** file and **images** folder (No need to open it within an IDE)
2. Double-click the **HuntTheWumpus.exe** file

# Reflection

What did you learn from this project.