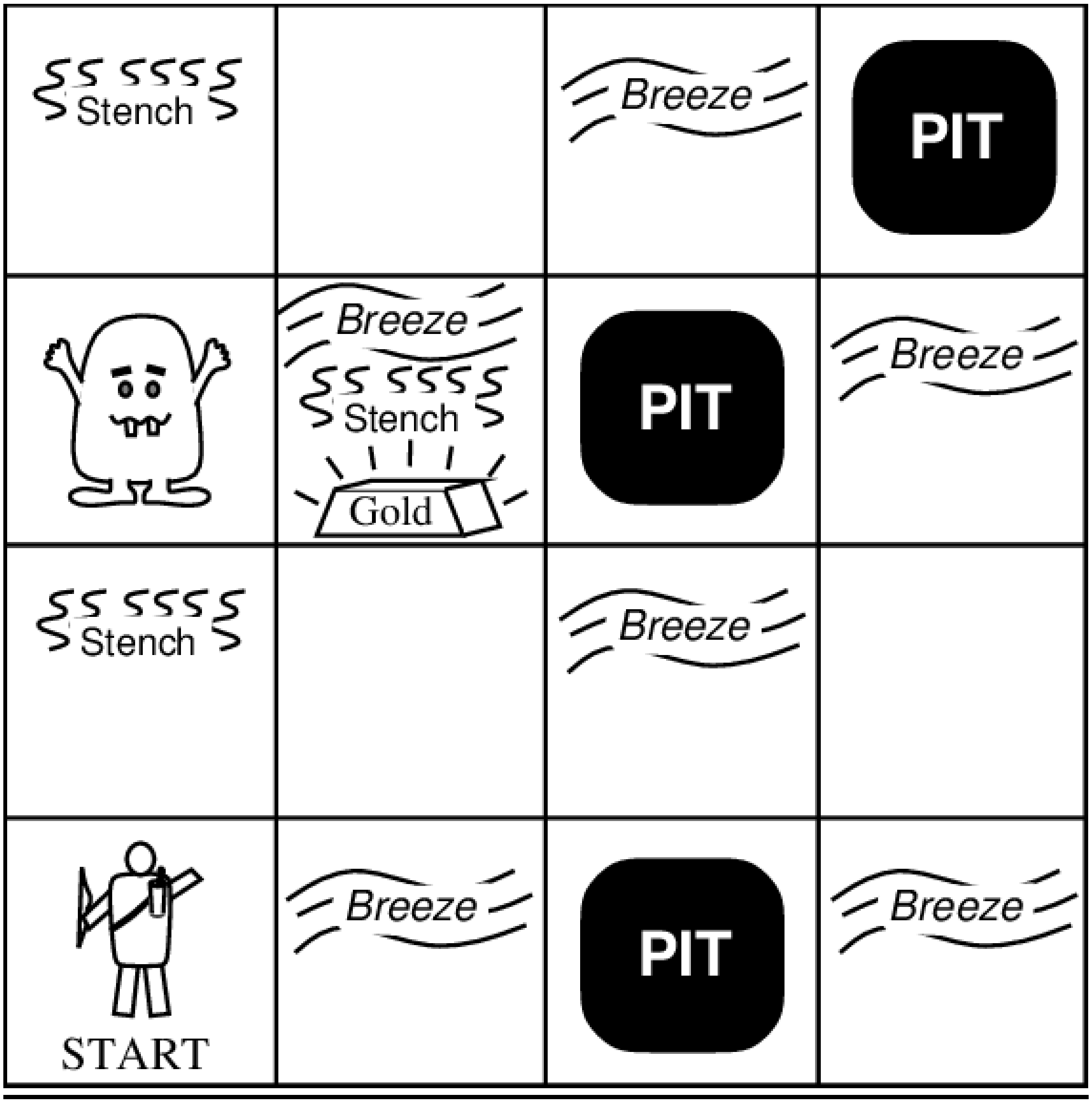
Al-alamein international university

AI (Artificial intelligence)

Dr.Layla shokry

Wampus world game



Team members

Abdelrahman elsawy	22100563
Hassan wael	22100274
Youssif Yasser	22100809
Hassan saleh	22101338
Mohammed ahmed	22101115

The parts of the game (description)

Wampus World is a popular exercise in the field of artificial intelligence and computer science. It is a simple "world" in which an "agent" operates, making decisions based on the information it gathers through "percepts." The agent's goal is to find gold and avoid encountering a dangerous creature called the Wumpus.

Here's a breakdown of the key components of a Wampus World project:

1. Knowledge base: The knowledge base is a central component of the Wampus World project. It contains information about the state of the world, including the locations of the gold, the Wumpus, pits, and the agent itself. This knowledge is used by the agent to make informed decisions about its actions.

2. Inference engine: The inference engine is responsible for processing the information in the knowledge base and using it to make logical deductions about the state of the world. It allows the agent to reason about its environment and make decisions based on that reasoning.

3. Agent function: The agent function determines how the agent behaves in the Wampus World. It specifies the rules that the agent follows to make decisions based on its percepts and the information in the knowledge base. The agent function is an essential part of the project, as it governs the behavior of the agent as it navigates the world.

4. Percepts: Percepts are the sensory input that the agent receives from the environment. In the case of Wampus World, percepts might include information about the agent's current location, the presence of a breeze (indicating a pit nearby), a stench (indicating the presence of the Wumpus), or the glint of gold. The agent uses these percepts to update its knowledge base and make decisions about its next actions.

5. Score updates: In a Wampus World project, the agent's performance is often measured by a score that reflects its success in finding gold and avoiding danger. The score is updated based on the agent's actions and the outcomes of those actions. For example, the agent might receive points for picking up gold and lose points for falling into a pit or encountering the Wumpus.

6. Visualization: Visualization is an important aspect of a Wampus World project, as it allows for the graphical representation of the agent's actions and the state of the world. Visualization can help researchers and developers understand the agent's behavior and assess its performance. It can also be useful for educational purposes, as it provides a more intuitive understanding of the project.

7.GUI(graphical user interface): a simple Tkinter-based GUI is created with buttons for the agent's actions (move, turn, grab, shoot). When each button is clicked, its corresponding method is called, which would trigger the game logic to perform the action in the Wumpus World game.

the actual game logic, including the visual representation of the game board, agent, wumpus, pits, and gold, would need to be implemented within the move, turn, grab, and shoot methods. Additionally, the game logic could use a separate class to represent the game state and perform the necessary computations.

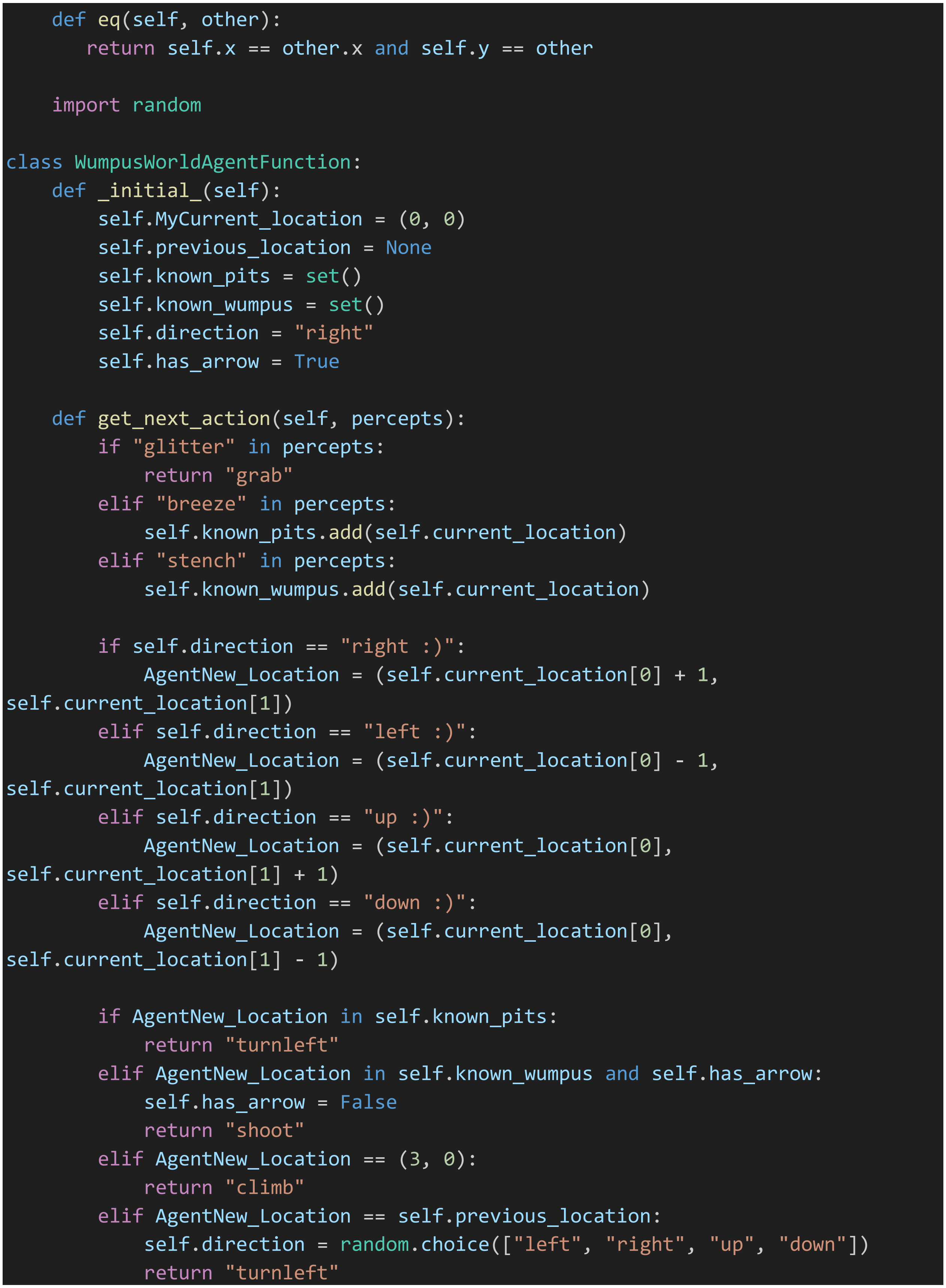
This example provides a basic structure for creating a GUI for a Wumpus World game in Python, and it can be expanded upon to include additional features such as displaying the game state, updating the state based on the agent's actions, and handling user interactions.

Conclusion

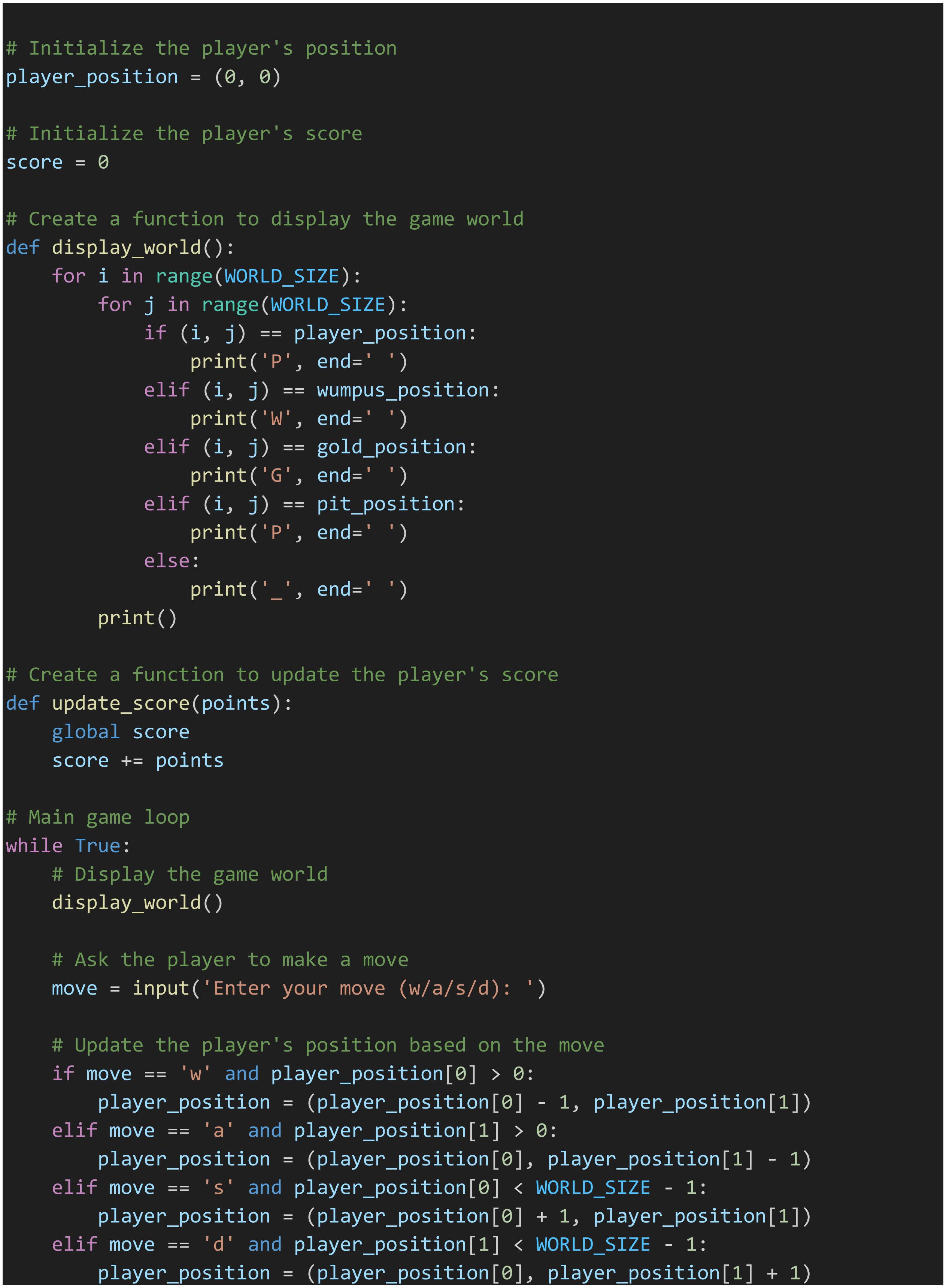
Overall, a Wampus World project typically involves creating a simulation of the world, implementing an agent with a knowledge base, inference engine, and agent function, and visualizing the agent's actions and the state of the world. This project allows for the exploration of fundamental concepts in artificial intelligence, such as logical reasoning, decision-making, and agent-based systems.

Code

```
import random

class WumpusGame:
    def _init_(self, size=5, num_pits=3, num_levels=3, max_arrows=3):
        self.size = size
        self.num_pits = num_pits
        self.num_levels = num_levels
        self.max_arrows = max_arrows
        self.current_level = 1
        self.player_position = (0, 0)
        self.wumpus_position = self.generate_random_position()
        self.gold_position = self.generate_random_position()
        self.pit_positions = [self.generate_random_position() for _ in
range(self.num_pits)]
        self.arrow_positions = []
        self.is_game_over = False
        self.inventory = []
        self.score = 0

    def generate_random_position(self):
        return random.randint(0, self.size - 1), random.randint(0, self.size - 1)

    def print_board(self):
        print(f"Level: {self.current_level} | Arrows: {self.max_arrows} | Score:
{self.score}")
        for i in range(self.size):
            for j in range(self.size):
                if (i, j) == self.player_position:
                    print("P", end=" ")
                elif (i, j) == self.wumpus_position:
                    print("W", end=" ")
                elif (i, j) == self.gold_position:
                    print("G", end=" ")
                elif (i, j) in self.pit_positions:
                    print("Pit", end=" ")
                elif (i, j) in self.arrow_positions:
                    print("A", end=" ")
                else:
                    print(".", end=" ")
            print()

    def check_encounter(self):
        if self.player_position == self.wumpus_position:
            print("You were eaten by the Wumpus! Game over.")
            self.is_game_over = True
        elif self.player_position == self.gold_position:
            print("Congratulations! You found the gold. You win!")
            self.inventory.append("Gold")
            self.score += 100
            self.advance_to_next_level()
        elif self.player_position in self.pit_positions:
            print("You fell into a pit! Game over.")
            self.is_game_over = True

    def advance_to_next_level(self):
        if self.current_level < self.num_levels:
            print("Advancing to the next level...")
            self.current_level += 1
            self.player_position = (0, 0)
            self.wumpus_position = self.generate_random_position()
            self.gold_position = self.generate_random_position()
            self.pit_positions = [self.generate_random_position() for _ in
range(self.num_pits)]
            self.arrow_positions = []
        else:
            print("Congratulations! You have completed all levels.")
            self.is_game_over = True

    def move_player(self, direction):
        x, y = self.player_position
        if direction == "UP" and x > 0:
            x -= 1
        elif direction == "DOWN" and x < self.size - 1:
            x += 1
        elif direction == "LEFT" and y > 0:
            y -= 1
        elif direction == "RIGHT" and y < self.size - 1:
            y += 1

        self.player_position = (x, y)
        self.check_encounter()
        if not self.is_game_over:
            print("Current position:")
            self.print_board()
            self.check_hints()

    def shoot_arrow(self, direction):
        if self.max_arrows <= 0:
            print("You are out of arrows!")
            return

        x, y = self.player_position
        self.arrow_positions.append(self.player_position)

        while True:
            if direction == "UP" and x > 0:
                x -= 1
            elif direction == "DOWN" and x < self.size - 1:
                x += 1
            elif direction == "LEFT" and y > 0:
                y -= 1
            elif direction == "RIGHT" and y < self.size - 1:
                y += 1
            else:
                break

            if (x, y) == self.wumpus_position:
                print("You shot the Wumpus! Congratulations!")
                self.inventory.append("Wumpus")
                self.wumpus_position = self.generate_random_position()
                self.score += 50
                break
            elif (x, y) in self.pit_positions:
                print("Arrow missed and hit a pit. Be careful!")
                break

        self.max_arrows -= 1
        self.check_encounter()

    def check_hints(self):
        x, y = self.player_position

        # Check for nearby dangers
        if (x - 1, y) == self.wumpus_position or (x + 1, y) ==
self.wumpus_position or \
           (x, y - 1) == self.wumpus_position or (x, y + 1) ==
self.wumpus_position:
            print("You smell a terrible odor. The Wumpus might be nearby!")

        if (x - 1, y) in self.pit_positions or (x + 1, y) in self.pit_positions
or \
           (x, y - 1) in self.pit_positions or (x, y + 1) in self.pit_positions:
            print("You feel a breeze. There might be a pit nearby.")

    def play_game(self):
        print("Welcome to Wumpus World!")
        print("Avoid the Wumpus, find the gold, and don't fall into pits.")
        print("Shoot the Wumpus with limited arrows to earn extra points.")

        while not self.is_game_over:
            self.print_board()
            action = input("Enter your move (MOVE/SHOOT): ").upper()

            if action == "MOVE":
                direction = input("Enter your move direction
(UP/DOWN/LEFT/RIGHT): ").upper()
                self.move_player(direction)
            elif action == "SHOOT":
                direction = input("Enter arrow direction (UP/DOWN/LEFT/RIGHT):
").upper()
                self.shoot_arrow(direction)
            else:
                print("Invalid action. Please enter MOVE or SHOOT.")

import random

class Percept(object):
    def _init_(self):
        self.stench = False
        self.breeze = False
        self.glitter = False
        self.bump = False
        self.scream = False

    def initialize(self):
        self.stench = False
        self.breeze = False
        self.glitter = False
        self.bump = False
        self.scream = False

class Game(object):
    def _init_(self):
        if file_information is None:
            # Set up game randomly.
            self.enemy_location = self._get_enemy_location()
            self.obstacle_locations = self._get_obstacle_locations()
        else:
            # Use file information for game setup.
            self.enemy_location = file_information.enemy_location
            self.obstacle_locations = file_information.obstacle_locations

        # Set initial player location and other game aspects.
        self.player_location = Location(1, 1)

    def initialize(self):
        # Reset game aspects back to default at the start of a new try.
        self.player_location = Location(1, 1)

    def _get_enemy_location(self):
        x, y = self._get_random_location()
        return Location(x, y)

    @staticmethod
    def _get_random_location():
        x = 1
        y = 1

        while (x == 1) and (y == 1):
            x = random.randint(1, WORLD_SIZE)
            y = random.randint(1, WORLD_SIZE)



    @staticmethod
    def _get_obstacle_locations():
       locations = []

       for x in range(1, WORLD_SIZE + 1):
           for y in range(1, WORLD_SIZE + 1):
               if (x != 1) or (y != 1):
                   if (random.randint(0, 1000 - 1)) < (OBSTACLE_PROBABILITY *
1000):
                       locations.append(Location(x, y))

       return locations

class Location(object):
    def _init_(self, x=0, y=0):
       self.x = x
       self.y = y
```

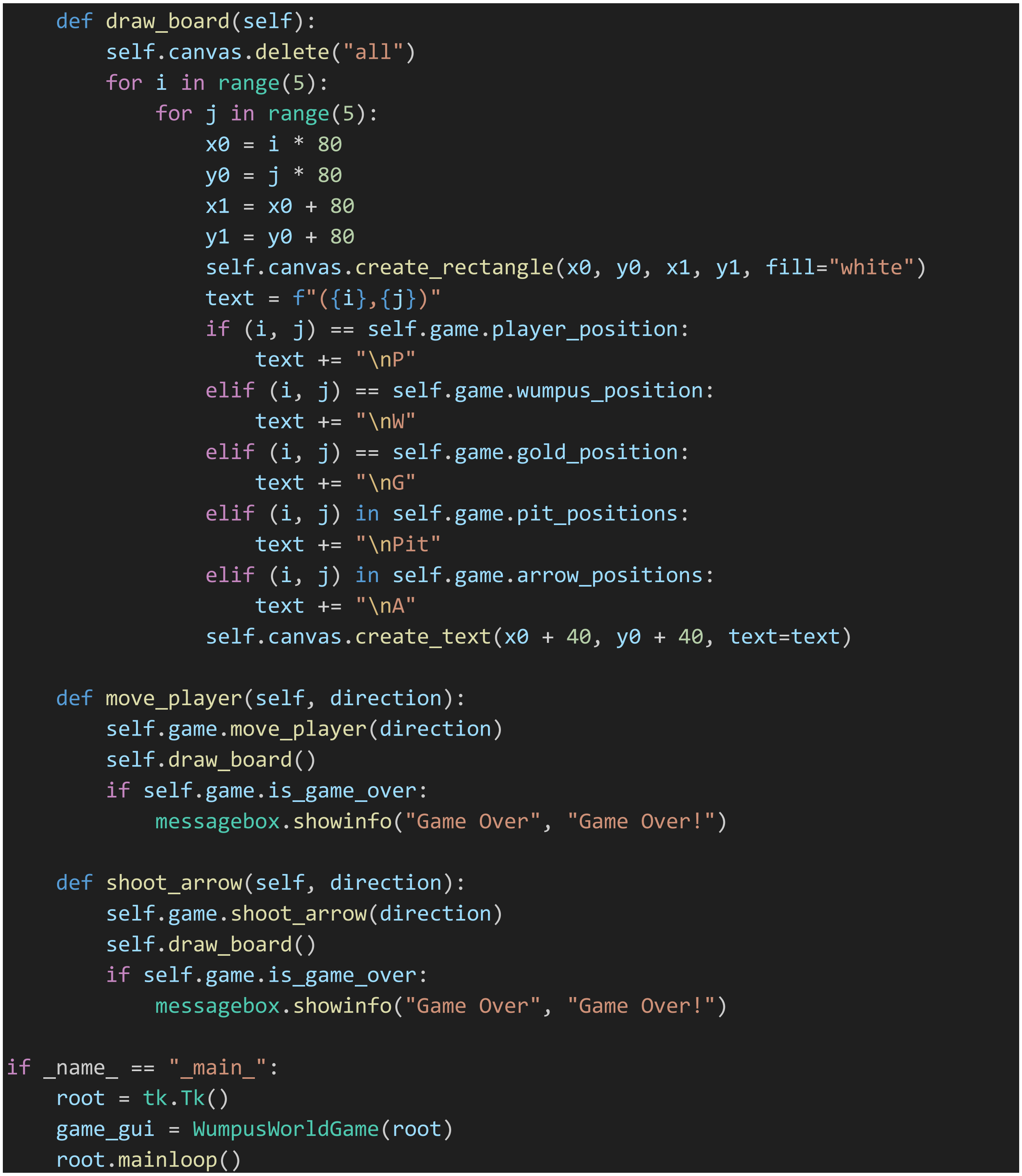


```
    def eq(self, other):
       return self.x == other.x and self.y == other

    import random

class WumpusWorldAgentFunction:
    def _initial_(self):
        self.MyCurrent_location = (0, 0)
        self.previous_location = None
        self.known_pits = set()
        self.known_wumpus = set()
        self.direction = "right"
        self.has_arrow = True

    def get_next_action(self, percepts):
        if "glitter" in percepts:
            return "grab"
        elif "breeze" in percepts:
            self.known_pits.add(self.current_location)
        elif "stench" in percepts:
            self.known_wumpus.add(self.current_location)

        if self.direction == "right :)":
            AgentNew_Location = (self.current_location[0] + 1,
self.current_location[1])
        elif self.direction == "left :)":
            AgentNew_Location = (self.current_location[0] - 1,
self.current_location[1])
        elif self.direction == "up :)":
            AgentNew_Location = (self.current_location[0],
self.current_location[1] + 1)
        elif self.direction == "down :)":
            AgentNew_Location = (self.current_location[0],
self.current_location[1] - 1)

        if AgentNew_Location in self.known_pits:
            return "turnleft"
        elif AgentNew_Location in self.known_wumpus and self.has_arrow:
            self.has_arrow = False
            return "shoot"
        elif AgentNew_Location == (3, 0):
            return "climb"
        elif AgentNew_Location == self.previous_location:
            self.direction = random.choice(["left", "right", "up", "down"])
            return "turnleft"
        else:
            self.previous_location = self.current_location
            self.current_location = AgentNew_Location
            return "forward"

        class InferenceEngine:
          def _init_(self, knowledge_base):
           self.knowledge_base = knowledge_base

    def make_inference(self, query):
        return self.knowledge_base.ask(query)


class KnowledgeBase:
    def _init_(self, facts):
        self.facts = facts

    def tell(self, new_fact):
        self.facts.append(new_fact)

    def ask(self, query):
        return query in self.facts


# Example usage:

    # Initialize the knowledge base with some initial facts
    initial_facts = ["Breeze in (1,2)", "Pit in (1,2)", "Stench in (2,1)",
"Wumpus in (3,3)"]


    # Make an inference

import random

# Define the size of the game world
WORLD_SIZE = 5

# Initialize the game world with random positions for the Wumpus, gold, and pit
wumpus_position = (random.randint(0, WORLD_SIZE - 1), random.randint(0,
WORLD_SIZE - 1))
gold_position = (random.randint(0, WORLD_SIZE - 1), random.randint(0, WORLD_SIZE
- 1))
pit_position = (random.randint(0, WORLD_SIZE - 1), random.randint(0, WORLD_SIZE -
1))
```



```python
# Initialize the player's position
player_position = (0, 0)

# Initialize the player's score
score = 0

# Create a function to display the game world
def display_world():
    for i in range(WORLD_SIZE):
        for j in range(WORLD_SIZE):
            if (i, j) == player_position:
                print('P', end=' ')
            elif (i, j) == wumpus_position:
                print('W', end=' ')
            elif (i, j) == gold_position:
                print('G', end=' ')
            elif (i, j) == pit_position:
                print('P', end=' ')
            else:
                print('_', end=' ')
        print()

# Create a function to update the player's score
def update_score(points):
    global score
    score += points

# Main game loop
while True:
    # Display the game world
    display_world()

    # Ask the player to make a move
    move = input('Enter your move (w/a/s/d): ')

    # Update the player's position based on the move
    if move == 'w' and player_position[0] > 0:
        player_position = (player_position[0] - 1, player_position[1])
    elif move == 'a' and player_position[1] > 0:
        player_position = (player_position[0], player_position[1] - 1)
    elif move == 's' and player_position[0] < WORLD_SIZE - 1:
        player_position = (player_position[0] + 1, player_position[1])
    elif move == 'd' and player_position[1] < WORLD_SIZE - 1:
        player_position = (player_position[0], player_position[1] + 1)
    # Check for encounters with the Wumpus, gold, or pit
    if player_position == wumpus_position:
        update_score(-100)
        print('You encountered the Wumpus! Score -100')
        break
    elif player_position == gold_position:
        update_score(100)
        print('You found the gold! Score +100')
        break
    elif player_position == pit_position:
        update_score(-50)
        print('You fell into a pit! Score -50')
        break

# Display the final score
print('Final score:', score)

import tkinter as tk
from tkinter import messagebox
import random

class WumpusGame:
    def _init_(self):
        self.size = 5
        self.num_pits = 3
        self.num_levels = 3
        self.max_arrows = 3
        self.current_level = 1
        self.player_position = (0, 0)
        self.wumpus_position = self.generate_random_position()
        self.gold_position = self.generate_random_position()
        self.pit_positions = [self.generate_random_position() for _ in
range(self.num_pits)]
        self.arrow_positions = []
        self.is_game_over = False
        self.inventory = []
        self.score = 0

    def generate_random_position(self):
        return random.randint(0, self.size - 1), random.randint(0, self.size - 1)

    def move_player(self, direction):
        x, y = self.player_position
        if direction == "UP" and x > 0:
            x -= 1
        elif direction == "DOWN" and x < self.size - 1:
            x += 1
        elif direction == "LEFT" and y > 0:
            y -= 1
        elif direction == "RIGHT" and y < self.size - 1:
            y += 1

        self.player_position = (x, y)
        self.check_encounter()
        if not self.is_game_over:
            self.check_hints()

    def shoot_arrow(self, direction):
        if self.max_arrows <= 0:
            messagebox.showinfo("Out of Arrows", "You are out of arrows!")
            return

        x, y = self.player_position
        self.arrow_positions.append(self.player_position)

        while True:
            if direction == "UP" and x > 0:
                x -= 1
            elif direction == "DOWN" and x < self.size - 1:
                x += 1
            elif direction == "LEFT" and y > 0:
                y -= 1
            elif direction == "RIGHT" and y < self.size - 1:
                y += 1
            else:
                break

            if (x, y) == self.wumpus_position:
                messagebox.showinfo("Wumpus Shot", "You shot the Wumpus!
Congratulations!")
                self.inventory.append("Wumpus")
                self.wumpus_position = self.generate_random_position()
                self.score += 50
                break
            elif (x, y) in self.pit_positions:
                messagebox.showinfo("Arrow Missed", "Arrow missed and hit a pit.
Be careful!")
                break
        self.max_arrows -= 1
        self.check_encounter()

    def check_encounter(self):
        if self.player_position == self.wumpus_position:
            messagebox.showinfo("Game Over", "You were eaten by the Wumpus! Game
over.")
            self.is_game_over = True
        elif self.player_position == self.gold_position:
            messagebox.showinfo("Congratulations", "You found the gold. You
win!")
            self.inventory.append("Gold")
            self.score += 100
            self.advance_to_next_level()
        elif self.player_position in self.pit_positions:
            messagebox.showinfo("Game Over", "You fell into a pit! Game over.")
            self.is_game_over = True

    def advance_to_next_level(self):
        if self.current_level < self.num_levels:
            messagebox.showinfo("Level Up", "Advancing to the next level...")
            self.current_level += 1
            self.player_position = (0, 0)
            self.wumpus_position = self.generate_random_position()
            self.gold_position = self.generate_random_position()
            self.pit_positions = [self.generate_random_position() for _ in
range(self.num_pits)]
            self.arrow_positions = []
        else:
            messagebox.showinfo("Game Over", "Congratulations! You have completed
all levels.")
            self.is_game_over = True

    def check_hints(self):
        x, y = self.player_position

        # Check for nearby dangers
        if (x - 1, y) == self.wumpus_position or (x + 1, y) ==
self.wumpus_position or \
           (x, y - 1) == self.wumpus_position or (x, y + 1) ==
self.wumpus_position:
            messagebox.showinfo("Hint", "You smell a terrible odor. The Wumpus
might be nearby!")
```

```
        if (x - 1, y) in self.pit_positions or (x + 1, y) in self.pit_positions
or \
           (x, y - 1) in self.pit_positions or (x, y + 1) in self.pit_positions:
            messagebox.showinfo("Hint", "You feel a breeze. There might be a pit
nearby.")

class WumpusWorldGame:
    def _init_(self, master):
        self.master = master
        self.master.title("Wumpus World Game")

        self.game = WumpusGame()

        self.canvas = tk.Canvas(self.master, width=400, height=400)
        self.canvas.pack()

        self.draw_board()

        # Buttons for player movements and shooting
        move_button = tk.Button(self.master, text="Move UP", command=lambda:
self.move_player("UP"))
        move_button.pack(side=tk.LEFT)
        move_button = tk.Button(self.master, text="Move DOWN", command=lambda:
self.move_player("DOWN"))
        move_button.pack(side=tk.LEFT)
        move_button = tk.Button(self.master, text="Move LEFT", command=lambda:
self.move_player("LEFT"))
        move_button.pack(side=tk.LEFT)
        move_button = tk.Button(self.master, text="Move RIGHT", command=lambda:
self.move_player("RIGHT"))
        move_button.pack(side=tk.LEFT)

        shoot_button = tk.Button(self.master, text="Shoot UP", command=lambda:
self.shoot_arrow("UP"))
        shoot_button.pack(side=tk.LEFT)
        shoot_button = tk.Button(self.master, text="Shoot DOWN", command=lambda:
self.shoot_arrow("DOWN"))
        shoot_button.pack(side=tk.LEFT)
        shoot_button = tk.Button(self.master, text="Shoot LEFT", command=lambda:
self.shoot_arrow("LEFT"))
        shoot_button.pack(side=tk.LEFT)
        shoot_button = tk.Button(self.master, text="Shoot RIGHT", command=lambda:
self.shoot_arrow("RIGHT"))
        shoot_button.pack(side=tk.LEFT)
```



```
    def draw_board(self):
        self.canvas.delete("all")
        for i in range(5):
            for j in range(5):
                x0 = i * 80
                y0 = j * 80
                x1 = x0 + 80
                y1 = y0 + 80
                self.canvas.create_rectangle(x0, y0, x1, y1, fill="white")
                text = f"({i},{j})"
                if (i, j) == self.game.player_position:
                    text += "\nP"
                elif (i, j) == self.game.wumpus_position:
                    text += "\nW"
                elif (i, j) == self.game.gold_position:
                    text += "\nG"
                elif (i, j) in self.game.pit_positions:
                    text += "\nPit"
                elif (i, j) in self.game.arrow_positions:
                    text += "\nA"
                self.canvas.create_text(x0 + 40, y0 + 40, text=text)

    def move_player(self, direction):
        self.game.move_player(direction)
        self.draw_board()
        if self.game.is_game_over:
            messagebox.showinfo("Game Over", "Game Over!")

    def shoot_arrow(self, direction):
        self.game.shoot_arrow(direction)
        self.draw_board()
        if self.game.is_game_over:
            messagebox.showinfo("Game Over", "Game Over!")

if _name_ == "_main_":
    root = tk.Tk()
    game_gui = WumpusWorldGame(root)
    root.mainloop()
```