NAME:AbdulRhman Ahmed Mohamed ID:22101286
.....rest of team member

```
import tkinter as tk
from tkinter import messagebox
import random


class KnowledgeBase:
    def __init__(self, size):
        self.size = size
        self.rooms = set()
        self.wumpus = None
        self.gold = None
        self.halls = set()

    def initialize(self):
        self._generate_rooms()
        self._generate_wumpus()
        self._generate_gold()
        self._generate_halls()

    def _generate_rooms(self):
        self.rooms = {(x, y) for x in range(1, self.size + 1) for y in
range(1, self.size + 1)}

    def _generate_wumpus(self):
        self.wumpus = random.sample(list(self.rooms), 1)[0]

    def _generate_gold(self):
        self.gold = random.sample(list(self.rooms - {self.wumpus}), 1)[0]

    def _generate_halls(self):
        self.halls = random.sample(list(self.rooms - {self.wumpus,
self.gold}), 2)

    def has_wumpus(self, x, y):
        return (x, y) == self.wumpus

    def has_gold(self, x, y):
        return (x, y) == self.gold

    def is_hall(self, x, y):
        return (x, y) in self.halls


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

    def infer(self, x, y):
        if self.knowledge_base.has_wumpus(x, y):
            return "You have hunted by the wumpus"

        if self.knowledge_base.has_gold(x, y):
            return "You see a glimmer of gold."

        if self.knowledge_base.is_hall(x, y):
            return "You fell into a pit! Score -500."
```

```
        return "The air is fresh."

    def update_knowledge_base(self, x, y):
        self.knowledge_base.wumpus = None
        self.knowledge_base.gold = None

    def is_adjacent_to_wumpus(self, x, y):
        wumpus_x, wumpus_y = self.knowledge_base.wumpus
        return (
                (x == wumpus_x and abs(y - wumpus_y) == 1) or
                (y == wumpus_y and abs(x - wumpus_x) == 1)
        )


class Agent:
    def __init__(self, knowledge_base, inference_engine, canvas, cell_size):
        self.knowledge_base = knowledge_base
        self.inference_engine = inference_engine
        self.canvas = canvas
        self.cell_size = cell_size
        self.current_position = (1, 1)
        self.score = 0
        self.move_count = 0
        self.max_moves = 4

        self.agent_id = canvas.create_text(
            1 * cell_size - cell_size / 2,
            (knowledge_base.size - 1) * cell_size + cell_size / 2,
            text="□"
        )

        self.gold_id = canvas.create_text(
            knowledge_base.gold[0] * cell_size - cell_size / 2,
            (knowledge_base.size - knowledge_base.gold[1]) * cell_size +
cell_size / 2,
            text="💰"
        )

        self.wumpus_id = canvas.create_text(
            knowledge_base.wumpus[0] * cell_size - cell_size / 2,
            (knowledge_base.size - knowledge_base.wumpus[1]) * cell_size +
cell_size / 2,
            text="👾"
        )

        self.hall_ids = []
        for hall in knowledge_base.halls:
            hall_id = canvas.create_text(
                hall[0] * cell_size - cell_size / 2,
                (knowledge_base.size - hall[1]) * cell_size + cell_size / 2,
                text="🕳"
            )
            self.hall_ids.append(hall_id)

    def move(self, direction):
        if self.move_count >= self.max_moves or not
```

```
self._is_valid_move(direction):
            messagebox.showinfo("Invalid Move", f"You cannot move further
{direction}.")
            return False

        x, y = self.current_position
        if direction == "up":
            y += 1
        elif direction == "down":
            y -= 1
        elif direction == "right":
            x += 1
        elif direction == "left":
            x -= 1

        self.canvas.move(self.agent_id, (x - self.current_position[0]) *
self.cell_size,
                         - (y - self.current_position[1]) * self.cell_size)
        self.current_position = (x, y)
        self.move_count += 1
        self.update_perceptions()
        self.update_score(-1)
        return True

    def _is_valid_move(self, direction):
        x, y = self.current_position
        return (
                (direction == "up" and y < self.knowledge_base.size) or
                (direction == "down" and y > 1) or
                (direction == "right" and x < self.knowledge_base.size) or
                (direction == "left" and x > 1)
        )

    def update_perceptions(self):
        x, y = self.current_position
        message = self.inference_engine.infer(x, y)

        if "Score -500" in message:
            self.fall_in_hall()
        else:
            # Check if the agent is adjacent to the Wumpus
            if self.inference_engine.is_adjacent_to_wumpus(x, y):
                messagebox.showinfo("Perception", "You smell a terrible odor!
Wumpus nearby!")
            else:
                messagebox.showinfo("Perception", message)

    def fall_in_hall(self):
        messagebox.showinfo("Fell in a Pit", "You fell into a pit! Score -
500.")
        self.climb()

    def grab(self):
        x, y = self.current_position
        if self.knowledge_base.has_gold(x, y):
            messagebox.showinfo("Grab Gold", "You have successfully grabbed
the gold!")
```

```
            self.knowledge_base.gold = None
            self.canvas.delete(self.gold_id)
            self.update_score(1000)
            return True
        else:
            messagebox.showinfo("No Gold", "There is no gold to grab.")
            return False

    def climb(self):
        x, y = self.current_position
        messagebox.showinfo("Climb", "Congratulations! You have climbed out
of the hall")

    def update_score(self, points):
        self.score += points
        messagebox.showinfo("Score Update", f"Current Score: {self.score}")


class WumpusWorldGUI:
    def __init__(self, size):
        self.size = size
        self.knowledge_base = KnowledgeBase(size)
        self.knowledge_base.initialize()

        self.inference_engine = InferenceEngine(self.knowledge_base)

        self.window = tk.Tk()
        self.window.title("Wumpus World Game")

        self.canvas = tk.Canvas(self.window, width=size * 50, height=size *
50)
        self.canvas.pack()

        self.cell_size = 50

        self.agent = Agent(self.knowledge_base, self.inference_engine,
self.canvas, self.cell_size)
        self.agent.current_position = (1, 1)

        self.create_grid()
        self.create_buttons()

    def create_grid(self):
        for i in range(1, self.size + 1):
            for j in range(1, self.size + 1):
                x1 = (i - 1) * self.cell_size
                y1 = (self.size - j) * self.cell_size
                x2 = i * self.cell_size
                y2 = (self.size - j + 1) * self.cell_size
                self.canvas.create_rectangle(x1, y1, x2, y2, outline="black")

    def create_buttons(self):
        button_frame = tk.Frame(self.window)
        button_frame.pack()

        move_button_up = tk.Button(button_frame, text="Move Up",
command=lambda: self.agent.move("up"))
```

```
        move_button_up.grid(row=0, column=1)

        move_button_down = tk.Button(button_frame, text="Move Down",
command=lambda: self.agent.move("down"))
        move_button_down.grid(row=1, column=1)

        move_button_left = tk.Button(button_frame, text="Move Left",
command=lambda: self.agent.move("left"))
        move_button_left.grid(row=1, column=0)

        move_button_right = tk.Button(button_frame, text="Move Right",
command=lambda: self.agent.move("right"))
        move_button_right.grid(row=1, column=2)

        grab_button = tk.Button(button_frame, text="Grab",
command=self.agent.grab)
        grab_button.grid(row=2, column=0)

        climb_button = tk.Button(button_frame, text="Climb",
command=self.agent.climb)
        climb_button.grid(row=2, column=2)

        score_button = tk.Button(button_frame, text="Score", command=lambda:
self.agent.update_score(0))
        score_button.grid(row=3, column=1)

    def run(self):
        self.window.mainloop()


if __name__ == "__main__":
    size = 4

    wumpus_game = WumpusWorldGUI(size)
    wumpus_game.run()
```