Faculty of Computing

[Artificial intelligence]

LAB#9

Name: Maira

Sapid: 45791

Batch: BSCS-6

Question 01:

Write a Python code using object-oriented classes to make a Simple Reflex Agent, as we have been doing in class. Below are the tasks that the agent must perform.

You manage a casino where a probabilistic game of wages is played. There are ‘n’ players and ‘n’ cards involved in this game. A card is mapped to a player based on the outcome of a roll of dice with-‘n’-faces. The problem is, you want to save as much money as you want therefore you want to fire your employee who hosts this game and rolls the dice. To save money, make an AI agent to replace the casino employee who performs the tasks below to host the game.

1. Identify number of the contestants
2. Add a same number of cards to the game
3. Perform the roll of two dice; one for players and the other for cards.
4. As per the value of the rolls, assign the corresponding card to the corresponding player.
5. Once a card is assigned to a players, both are invalid if the rolls of dice calls them again.
6. Announce the winner – the player with the highest card value, as per the legend below.

a. Cards Legend:

i. The bigger the number, the higher the priority

Spades > Hearts > Diamonds > Clubs

```
import random

class SimpleReflexAgent:
    def __init__(self):
        self.players = []
        self.cards = []
        self.assigned = {}  # {player: card}
        self.card_values = {
            'Spades': 4,
            'Hearts': 3,
            'Diamonds': 2,
            'Clubs': 1
        }

    def identify_contestants(self, n):
        """Step 1: Identify 'n' players."""
        self.players = [f"Player_{i+1}" for i in range(n)]
```

```
    def add_cards(self, n):
        """Step 2: Add 'n' cards (random suits and numbers)."""
        suits = ['Spades', 'Hearts', 'Diamonds', 'Clubs']
        self.cards = [
            (random.randint(1, 13), random.choice(suits))
            for _ in range(n)
        ]

    def roll_dice(self, n):
        """Step 3: Roll two n-faced dice (player and card selection)."""
        return random.randint(1, n), random.randint(1, n)

    def assign_cards(self):
        """Step 4 & 5: Assign cards to players via dice rolls (no
repeats)."""
        n = len(self.players)
        while len(self.assigned) < n:
            player_roll, card_roll = self.roll_dice(n)
            player = self.players[player_roll - 1]
            card = self.cards[card_roll - 1]

            if player not in self.assigned and card not in
self.assigned.values():
                self.assigned[player] = card

    def announce_winner(self):
        """Step 6: Announce winner (highest card per legend)."""
        max_score = -1
        winner = None

        for player, (number, suit) in self.assigned.items():
            score = number * 10 + self.card_values[suit]  # Priority: Number
> Suit
            if score > max_score:
                max_score = score
                winner = player

        print(f"\nAssigned Cards: {self.assigned}")
        print(f"Winner: {winner} with Card {self.assigned[winner]}")

# Run the game
if __name__ == "__main__":
    agent = SimpleReflexAgent()
    n = int(input("Enter number of players: "))
    agent.identify_contestants(n)
    agent.add_cards(n)
    agent.assign_cards()
    agent.announce_winner()
```

```
C:\Users\MASTERCOMPUTERS\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\MASTERCOMPUTERS\PycharmProjects\PythonProje
Enter number of players: 4

Assigned Cards: {'Player_2': (8, 'Spades'), 'Player_3': (4, 'Hearts'), 'Player_4': (7, 'Hearts'), 'Player_1': (1, 'Diamonds')}
Winner: Player_2 with Card (8, 'Spades')

Process finished with exit code 0
```

Question 02:

Write a program that includes the three different types of agents and their implementation in different scenarios.

- The program includes the implementation of goal-based agents.
- The program includes the implementation of the model-based agent.
- The program includes the implementation of a utility-based agent.

```
    class GoalBasedAgent:
    def __init__(self, goal):
        self.position = (0, 0)
        self.goal = goal

    def move(self):
        print(f"Current position: {self.position}")
        if self.position[0] < self.goal[0]:
            self.position = (self.position[0] + 1, self.position[1])
        elif self.position[1] < self.goal[1]:
            self.position = (self.position[0], self.position[1] + 1)

    def reached_goal(self):
        return self.position == self.goal

# Test Goal-Based Agent
print("\n Goal-Based Agent:")
goal_agent = GoalBasedAgent((2, 3))
while not goal_agent.reached_goal():
    goal_agent.move()
print(f"Reached goal at {goal_agent.position}")

class ModelBasedAgent:
    def __init__(self):
        self.position = (0, 0)
        self.environment_model = {(1, 0): "obstacle", (2, 2): "free"}

    def perceive(self, pos):
        return self.environment_model.get(pos, "unknown")

    def move_to(self, new_pos):
        if self.perceive(new_pos) != "obstacle":
            self.position = new_pos
            print(f"Moved to {self.position}")
        else:
            print(f"Blocked at {new_pos}!")

# Test Model-Based Agent
print("\n Model-Based Agent:")
model_agent = ModelBasedAgent()
model_agent.move_to((1, 0))  # Blocked
model_agent.move_to((1, 1))  # Allowed
```

```
model_agent.move_to((2, 2))  # Allowed

class UtilityBasedAgent:
    def __init__(self):
        self.choices = {
            "watch_movie": 5,
            "study": 8,
            "sleep": 6,
            "play_game": 7
        }

    def choose_best(self):
        best = max(self.choices, key=self.choices.get)
        print(f"Best choice based on utility: {best} (utility =
{self.choices[best]})")

# Test Utility-Based Agent
print("\n Utility-Based Agent:")
utility_agent = UtilityBasedAgent()
utility_agent.choose_best()
```

```
 Goal-Based Agent:
Current position: (0, 0)
Current position: (1, 0)
Current position: (2, 0)
Current position: (2, 1)
Current position: (2, 2)
Reached goal at (2, 3)

 Model-Based Agent:
Blocked at (1, 0)!
Moved to (1, 1)
Moved to (2, 2)

 Utility-Based Agent:
Best choice based on utility: study (utility = 8)

Process finished with exit code 0
```