RIPHAH INTERNATIONAL UNIVERSITY, ISLAMABAD

Lab#9

Bachelors of Computer Science – 6th Semester

Course: Artificial Intelligence

Submitted to: Ms. Ayesha

Submitted by: Tabinda Hassan

SAP-46374

Date of Submission: 11-04-2025

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

ii. Spades > Hearts > Diamonds > Clubs

```python
import random

class CasinoGame:
    def __init__(self, num_players):
        self.num_players = num_players
        self.players = list(range(1, num_players + 1))
        self.cards = self.create_deck()
        self.played_players = set()
        self.played_cards = set()

    def create_deck(self):
        suits = ["Spades", "Hearts", "Diamonds", "Clubs"]
        return [(value, suit) for value in range(2, 15) for suit in suits]

    def roll_dice(self, max_value):
        return random.randint(1, max_value)

    def assign_cards(self):
        for _ in range(self.num_players):
            while True:
                player_roll = self.roll_dice(self.num_players)
                card_roll = self.roll_dice(len(self.cards))
                if player_roll not in self.played_players and card_roll not in self.played_cards:
                    self.played_players.add(player_roll)
                    self.played_cards.add(card_roll)
                    player = f"Player {player_roll}"
                    card = self.cards[card_roll]
                    print(f"{player} got {card[0]} of {card[1]}")
                    break

    def announce_winner(self):
        def card_priority(card):
            suit_priority = {"Spades": 4, "Hearts": 3, "Diamonds": 2, "Clubs": 1}
            return card[0] * 10 + suit_priority[card[1]]

        winner = max(
            [(player, self.cards[card]) for player, card in zip(self.played_players, self.played_cards)],
            key=lambda pc: card_priority(pc[1])
        )
        print(f"The winner is Player {winner[0]} with {winner[1][0]} of {winner[1][1]}!")

    def play(self):
        print("Game starting...")
        self.assign_cards()
        self.announce_winner()

if __name__ == "__main__":
    num_players = int(input("Enter number of players: "))
    game = CasinoGame(num_players)
    game.play()

```

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.

```python
import random

class GoalBasedAgent:
    def achieve_goal(self, current_state, goal):
        if current_state < goal:
            return "Move forward"
        elif current_state == goal:
            return "Goal achieved"
        else:
            return "Backtrack"

class ModelBasedAgent:
    def __init__(self):
        self.environment = {"traffic_light": "red"}

    def update(self, traffic_light):
        self.environment["traffic_light"] = traffic_light

    def act(self):
        return "Move" if self.environment["traffic_light"] == "green" else "Wait"

class UtilityBasedAgent:
    def choose_action(self, options):
        utilities = {"fast": 3, "safe": 2, "cheap": 1}
        return max(options, key=lambda x: utilities.get(x, 0))
```

```python

if __name__ == "__main__":
    print("Goal-Based Agent:")
    goal_agent = GoalBasedAgent()
    print(goal_agent.achieve_goal(5, 10))

    print("\nModel-Based Agent:")
    model_agent = ModelBasedAgent()
    print(model_agent.act())
    model_agent.update("green")
    print(model_agent.act())

    print("\nUtility-Based Agent:")
    utility_agent = UtilityBasedAgent()
    print(utility_agent.choose_action(["fast", "cheap"]))

```