Artificial Intelligence

Lab 09 Tasks

Name: Samreen Bibi

Sap ID: 46484

Batch: BSCS-6th semester

Lab Instructor:

Mam Ayesha Akram

Task#1

Solution:

```
import random
class Card:  1 usage
    def __init__(self, number, suit):
        self.number = number
        self.suit = suit
    def get_value(self):  1 usage (1 dynamic)
        suit_priority = {"Clubs": 1, "Diamonds": 2, "Hearts": 3, "Spades": 4}
        return self.number * 10 + suit_priority[self.suit]
    def __str__(self):
        return f"{self.number} of {self.suit}"
class Player:  1 usage
    def __init__(self, id):
        self.id = id
        self.card = None
class CasinoAgent:  1 usage
    def __init__(self, num_players):
        self.num_players = num_players
        self.players = [Player(i+1) for i in range(num_players)]
        self.suits = ["Clubs", "Diamonds", "Hearts", "Spades"]
        self.cards = self.generate_cards()
        self.used_players = set()
        self.used_cards = set()
    def generate_cards(self):  1 usage
        cards = []
```

```
        for _ in range(self.num_players):
            number = random.randint( a: 1,  b: 13)
            suit = random.choice(self.suits)
            cards.append(Card(number, suit))
        return cards
    def play_game(self):  1 usage
        print("Starting game...\n")
        while len(self.used_players) < self.num_players:
            player_roll = random.randint( a: 1, self.num_players)
            card_roll = random.randint( a: 1, self.num_players)
            if player_roll in self.used_players or card_roll in self.used_cards:
                continue
            player = self.players[player_roll - 1]
            card = self.cards[card_roll - 1]
            player.card = card
            self.used_players.add(player_roll)
            self.used_cards.add(card_roll)
            print(f"Player {player.id} gets card: {card}")
        self.announce_winner()
    def announce_winner(self):  1 usage
        winner = max(self.players, key=lambda p: p.card.get_value())
        print(f"\nWinner is Player {winner.id} with card: {winner.card}")
num = int(input("Enter number of players: "))
agent = CasinoAgent(num)
agent.play_game()
```

Output:

```
Enter number of players: 5
Starting game...

Player 3 gets card: 13 of Spades
Player 2 gets card: 11 of Clubs
Player 1 gets card: 8 of Diamonds
Player 5 gets card: 6 of Spades
Player 4 gets card: 1 of Spades

Winner is Player 3 with card: 13 of Spades
```

Task#2

Solution:

```python
import random
class GoalBasedAgent:  1 usage
    def __init__(self, goal):
        self.goal = goal
    def act(self, environment):  1 usage
        if self.goal in environment:
            return f"Heading towards {self.goal}!"
        else:
            return f"Goal {self.goal} not found. Searching..."
class ModelBasedAgent:  1 usage
    def __init__(self):
        self.experience = {}
    def update_experience(self, action, result):  2 usages
        self.experience[action] = result

    def act(self):  1 usage
        if "danger" in self.experience:
            return "Action: Avoid the danger"
        else:
            return "Action: Keep moving forward"
class UtilityBasedAgent:  1 usage
    def __init__(self):
        pass
    def evaluate(self, action):  1 usage
        utility_scores = {"eat": 4, "study": 10, "play": 6}
        return utility_scores.get(action, 0)
```

```
    def act(self, actions):  1 usage
        best_action = max(actions, key=self.evaluate)
        return f"Selected Action: {best_action} (Best utility)"
print("Goal-Based Agent:")
goal_agent = GoalBasedAgent("Mountain")
environment = ["River", "Tree", "Mountain"]
print(goal_agent.act(environment))

print("\nModel-Based Agent:")
model_agent = ModelBasedAgent()
model_agent.update_experience( action: "path",  result: "clear")
model_agent.update_experience( action: "path",  result: "danger")
print(model_agent.act())

print("\nUtility-Based Agent:")
utility_agent = UtilityBasedAgent()
actions = ["eat", "study", "play"]
print(utility_agent.act(actions))
```

Output:

```
Goal-Based Agent:
Heading towards Mountain!

Model-Based Agent:
Action: Keep moving forward

Utility-Based Agent:
Selected Action: study (Best utility)
```