Artificial Intelligence Lab

Name: Hajira Imran

Sap ID: 44594

Batch: BSCS-6th semester

Instructor: Ayesh Akram

Task 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:

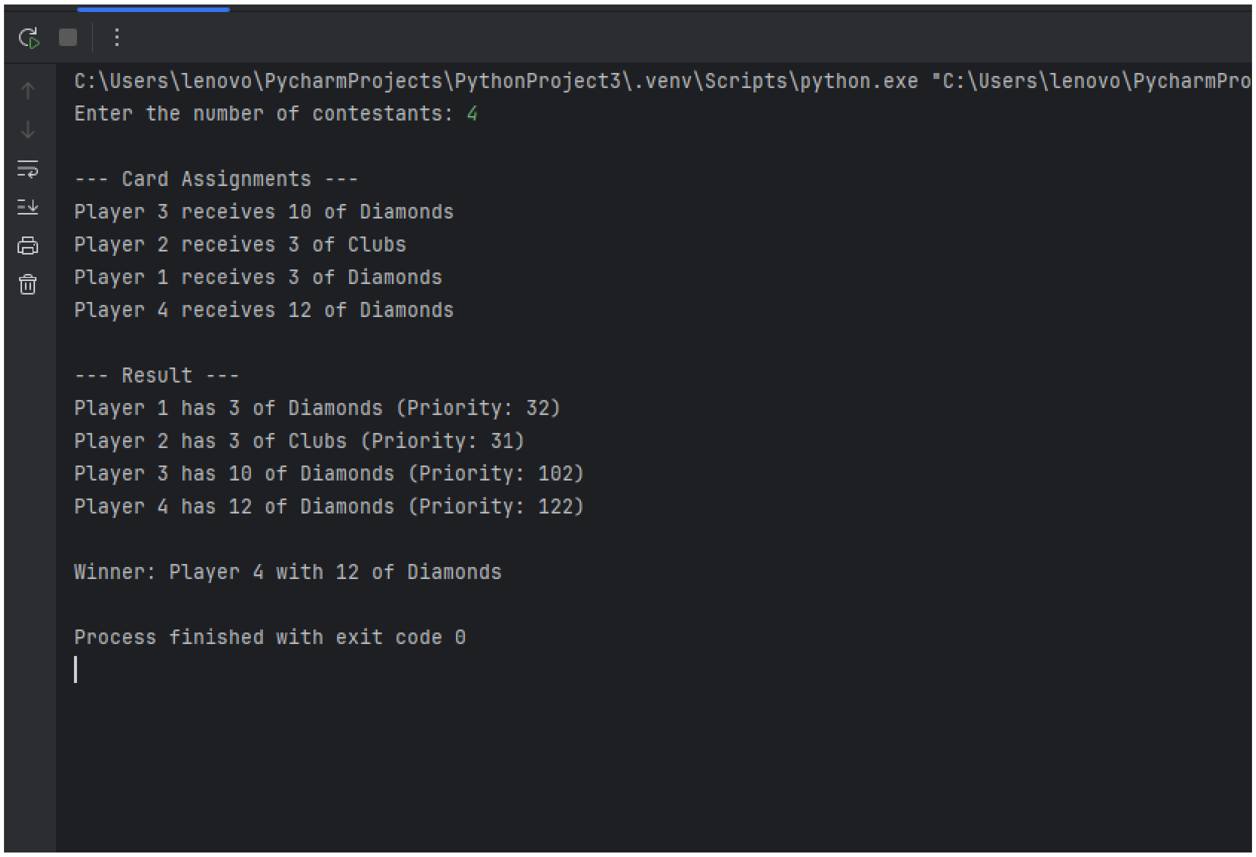
i. **The bigger the number, the higher the priority**

ii. **Spades > Hearts > Diamonds > Clubs**

Code

```
import random


# Define Card class
class Card:
    def __init__(self, value, suit):
        self.value = value
        self.suit = suit
        self.is_assigned = False

    def __str__(self):
        return f"{self.value} of {self.suit}"

    def get_priority(self):
        suit_priority = {'Clubs': 1, 'Diamonds': 2, 'Hearts': 3, 'Spades': 4}
        return self.value * 10 + suit_priority[self.suit]  # Higher number and higher suit has more
weight



# Define Player class
class Player:
    def __init__(self, id):
        self.id = id
        self.card = None
    def assign_card(self, card):
        self.card = card

    def __str__(self):
        return f"Player {self.id}"



# Define the Casino AI Agent
class CasinoAgent:
    def __init__(self, num_players):
        self.num_players = num_players
        self.players = [Player(i+1) for i in range(num_players)]
        self.cards = self.generate_cards()
        self.available_players = list(range(num_players))
        self.available_cards = list(range(num_players))

    def generate_cards(self):
        suits = ['Clubs', 'Diamonds', 'Hearts', 'Spades']
        cards = []
        for i in range(self.num_players):
            value = random.randint(1, 13)  # Card values from 1 to 13
            suit = random.choice(suits)
            cards.append(Card(value, suit))
        return cards

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


    def assign_cards_to_players(self):
        print("\n--- Card Assignments ---")
        while self.available_players and self.available_cards:
            player_roll = self.roll_dice() - 1
            card_roll = self.roll_dice() - 1


            if player_roll in self.available_players and card_roll in self.available_cards:
                player = self.players[player_roll]
                card = self.cards[card_roll]


                player.assign_card(card)
                card.is_assigned = True


                self.available_players.remove(player_roll)
                self.available_cards.remove(card_roll)


                print(f"{player} receives {card}")


    def announce_winner(self):
        print("\n--- Result ---")
        winner = None
        highest_priority = -1


        for player in self.players:
            if player.card:
                priority = player.card.get_priority()
                print(f"{player} has {player.card} (Priority: {priority})")
                if priority > highest_priority:
                    highest_priority = priority
                    winner = player

        if winner:
            print(f"\nWinner: {winner} with {winner.card}")
        else:
            print("No winner found.")

# Run the simulation
if __name__ == "__main__":
    n = int(input("Enter the number of contestants: "))
    agent = CasinoAgent(n)
    agent.assign_cards_to_players()
    agent.announce_winner()
```

Output



```
C:\Users\lenovo\PycharmProjects\PythonProject3\.venv\Scripts\python.exe "C:\Users\lenovo\PycharmPro
Enter the number of contestants: 4

--- Card Assignments ---
Player 3 receives 10 of Diamonds
Player 2 receives 3 of Clubs
Player 1 receives 3 of Diamonds
Player 4 receives 12 of Diamonds

--- Result ---
Player 1 has 3 of Diamonds (Priority: 32)
Player 2 has 3 of Clubs (Priority: 31)
Player 3 has 10 of Diamonds (Priority: 102)
Player 4 has 12 of Diamonds (Priority: 122)

Winner: Player 4 with 12 of Diamonds

Process finished with exit code 0
```

Task 02

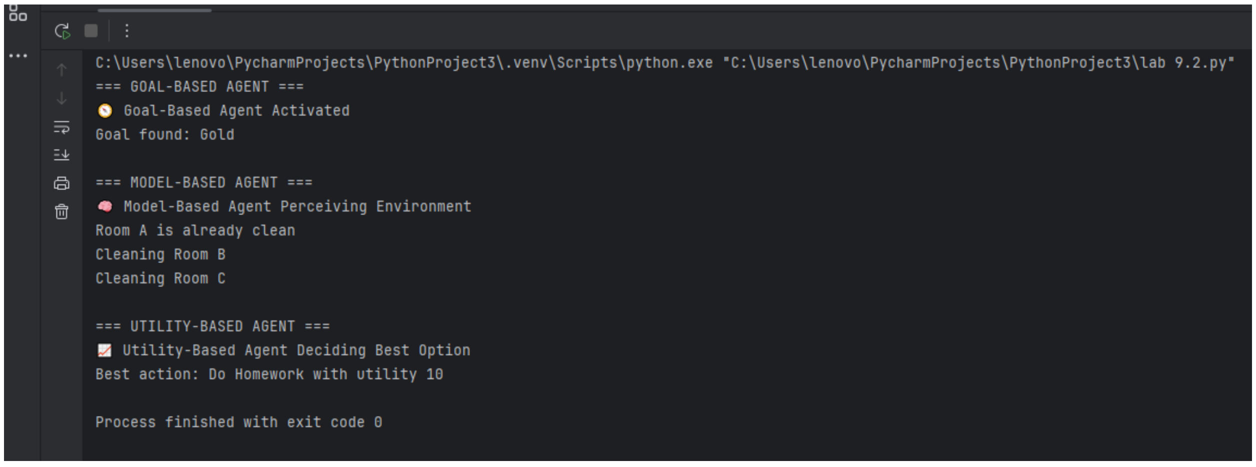
Write a program that includes the three different types of agents and their implementation in different scenarios.

- **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.**

Code

```
import random

# -----------------------
# 1. Goal-Based Agent
# -----------------------
class GoalBasedAgent:
    def __init__(self, goal):
        self.goal = goal

    def act(self, environment):
        print("Goal-Based Agent Activated")
        for item in environment:
            if item == self.goal:
                print(f"Goal found: {item}")
                return
        print("Goal not found.")



# -----------------------
# 2. Model-Based Agent
# -----------------------
class ModelBasedAgent:
    def __init__(self):
        self.internal_state = {}

    def perceive(self, environment):
        print(" Model-Based Agent Perceiving Environment")
        for location, status in environment.items():
            self.internal_state[location] = status

    def act(self):
        for loc, stat in self.internal_state.items():
            if stat == "dirty":
                print(f"Cleaning {loc}")
            else:
                print(f"{loc} is already clean")


# -----------------------
# 3. Utility-Based Agent
# -----------------------
class UtilityBasedAgent:
    def __init__(self, options):
        self.options = options  # options: [(action, utility)]

    def act(self):
        print(" Utility-Based Agent Deciding Best Option")
        best_action = max(self.options, key=lambda x: x[1])
        print(f"Best action: {best_action[0]} with utility {best_action[1]}")


# -----------------------
# MAIN FUNCTION TO TEST ALL
# -----------------------
def main():
    print("=== GOAL-BASED AGENT ===")
    goal_agent = GoalBasedAgent(goal="Gold")
    environment1 = ["Rock", "Sand", "Gold", "Stone"]
    goal_agent.act(environment1)

    print("\n=== MODEL-BASED AGENT ===")
    environment2 = {
        "Room A": "clean",
        "Room B": "dirty",
        "Room C": "dirty"
    }
    model_agent = ModelBasedAgent()
    model_agent.perceive(environment2)
    model_agent.act()

    print("\n=== UTILITY-BASED AGENT ===")
    actions = [
        ("Read Book", 5),
        ("Play Game", 2),
        ("Do Homework", 10),
        ("Watch Movie", 4)
    ]
    utility_agent = UtilityBasedAgent(actions)
    utility_agent.act()


main()
```

Output



```
C:\Users\lenovo\PycharmProjects\PythonProject3\.venv\Scripts\python.exe "C:\Users\lenovo\PycharmProjects\PythonProject3\lab 9.2.py"
=== GOAL-BASED AGENT ===
🧭 Goal-Based Agent Activated
Goal found: Gold

=== MODEL-BASED AGENT ===
🧠 Model-Based Agent Perceiving Environment
Room A is already clean
Cleaning Room B
Cleaning Room C

=== UTILITY-BASED AGENT ===
📈 Utility-Based Agent Deciding Best Option
Best action: Do Homework with utility 10

Process finished with exit code 0
```