Faculty of Computing

Artificial Intelligence

Lab # 9

Name: Alishba Waqar

Sap Id: 46997

BSCS-6

Instructor

Ayesha Akram,

TF, Faculty of Computing,

Riphah International University, Islamabad

LAB TASKS

Question 01

```python
import random

# Priority of card suits (higher is better)
SUIT_PRIORITY = {'Spades': 4, 'Hearts': 3, 'Diamonds': 2, 'Clubs': 1}

# Class for a single card
class Card:  1 usage
    def __init__(self, value, suit):
        self.value = value
        self.suit = suit

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

    def score(self):  2 usages (2 dynamic)
        return self.value * 10 + SUIT_PRIORITY[self.suit]

# Game environment to hold players and cards
class CasinoEnvironment:  1 usage
    def __init__(self, total_players):
        self.players = [f"Player {i+1}" for i in range(total_players)]
        self.cards = self.create_random_cards(total_players)
        self.assigned_cards = {}
```

```python
    def create_random_cards(self, count):  1 usage
        cards = []
        for _ in range(count):
            value = random.randint( a: 1,  b: 13)
            suit = random.choice(list(SUIT_PRIORITY.keys()))
            cards.append(Card(value, suit))
        return cards

# AI Agent to host the game
class CasinoAgent:  1 usage
    def __init__(self, environment):
        self.env = environment
        self.remaining_players = list(range(len(environment.players)))
        self.remaining_cards = list(range(len(environment.cards)))

    def roll_dice(self, sides):  2 usages
        return random.randint( a: 0, sides - 1)

    def assign_cards_to_players(self):  1 usage
        print("\nAssigning Cards to Players...\n")
        while self.remaining_players and self.remaining_cards:
            p_index = self.roll_dice(len(self.env.players))
            c_index = self.roll_dice(len(self.env.cards))
```

```
            if p_index in self.remaining_players and c_index in self.remaining_cards:
                player = self.env.players[p_index]
                card = self.env.cards[c_index]
                self.env.assigned_cards[player] = card
                print(f"{player} receives {card}")
                self.remaining_players.remove(p_index)
                self.remaining_cards.remove(c_index)
            else:
                print("Duplicate roll. Retrying...")

    def show_winner(self):  1 usage
        print("\nGame Result:\n")
        for player, card in self.env.assigned_cards.items():
            print(f"{player} → {card} (Score: {card.score()})")

        winner = max(self.env.assigned_cards.items(), key=lambda x: x[1].score())
        print(f"\nThe winner is: {winner[0]} with {winner[1]}!")

# Main Program
num = int(input("Enter number of players: "))
environment = CasinoEnvironment(num)
agent = CasinoAgent(environment)
```

```
# Main Program
num = int(input("Enter number of players: "))
environment = CasinoEnvironment(num)
agent = CasinoAgent(environment)

agent.assign_cards_to_players()
agent.show_winner()
```

```
Enter number of players: 2

Assigning Cards to Players...

Player 1 receives 8 of Hearts
Player 2 receives 11 of Hearts

Game Result:

Player 1 → 8 of Hearts (Score: 83)
Player 2 → 11 of Hearts (Score: 113)

The winner is: Player 2 with 11 of Hearts!
```

Question 02

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

    def search_goal(self, environment):  1 usage
        print("Goal-Based Agent is searching for a target item...")
        for item in environment:
            if item == self.goal:
                print(f"Target found: {item}")
                return
        print("Target not found.")
```

```python
# 2. Model-Based Agent
class ModelBasedAgent:  1 usage
    def __init__(self):
        self.memory = {}

    def update_memory(self, environment):  1 usage
        print("Model-Based Agent is observing areas...")
        for area, status in environment.items():
            self.memory[area] = status

    def perform_action(self):  1 usage
        for area, condition in self.memory.items():
            if condition == "occupied":
                print(f"Skipping {area} (occupied)")
            else:
                print(f"Entering {area}")
```

```python
# 3. Utility-Based Agent

class UtilityBasedAgent:  1 usage
    def __init__(self, activities):
        self.activities = activities  # List of (task, value)

    def get_utility(self, activity):  1 usage
        return activity[1]

    def choose_best(self):  1 usage
        print("Utility-Based Agent is selecting the best activity...")
        best = max(self.activities, key=self.get_utility)
        print(f"Selected activity: {best[0]} with utility {best[1]}")
```

```python
def main():  1 usage
    print("=== GOAL-BASED AGENT ===")
    goal_agent = GoalBasedAgent(goal="Treasure")
    environment_items = ["Tree", "Rock", "Water", "Treasure", "Bush"]
    goal_agent.search_goal(environment_items)

    print("\n=== MODEL-BASED AGENT ===")
    park_status = {
        "Zone A": "free",
        "Zone B": "occupied",
        "Zone C": "free",
        "Zone D": "occupied"
    }
    model_agent = ModelBasedAgent()
    model_agent.update_memory(park_status)
    model_agent.perform_action()

    print("\n=== UTILITY-BASED AGENT ===")
    activity_options = [
        ("Take a walk", 6),
        ("Watch TV", 4),
        ("Practice coding", 9),
        ("Scroll social media", 2)
    ]
    utility_agent = UtilityBasedAgent(activity_options)
    utility_agent.choose_best()
```

```
=== GOAL-BASED AGENT ===
Goal-Based Agent is searching for a target item...
Target found: Treasure

=== MODEL-BASED AGENT ===
Model-Based Agent is observing areas...
Entering Zone A
Skipping Zone B (occupied)
Entering Zone C
Skipping Zone D (occupied)

=== UTILITY-BASED AGENT ===
Utility-Based Agent is selecting the best activity...
Selected activity: Practice coding with utility 9
```