

Faculty of Computing



Artificial Intelligence

Spring 2025

Lab # 9

Name: Faareha Raza

Sap id: 47431

LAB TASKS

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

This screenshot shows the VS Code editor with the file `Lab9 q1.py` open. The code defines a `Card` class and a `Player` class. The `Card` class has attributes `value` and `suit`, and methods `__init__`, `__str__`, and `card_strength`. The `Player` class has attributes `id` and `card`, and methods `__init__` and `assign_card`. The `card_strength` method uses a `suit_priority` dictionary to determine the strength of a card based on its suit.

```
1 import random
2 suit_priority = {
3     "Spades": 4,
4     "Hearts": 3,
5     "Diamonds": 2,
6     "Clubs": 1
7 }
8 class Card:
9     def __init__(self, value, suit):
10         self.value = value
11         self.suit = suit
12
13     def __str__(self):
14         return f'{self.value} of {self.suit}'
15
16     def card_strength(self):
17         return (self.value, suit_priority[self.suit])
18 class Player:
19     def __init__(self, id):
20         self.id = id
21         self.card = None
22
23     def assign_card(self, card):
24         self.card = card
25
26     def __str__(self):
27         return f'Player {self.id}'
28 class CasinoAgent:
```

This screenshot shows the VS Code editor with the file `Lab9 q1.py` open, displaying the `CasinoAgent` class. The class has methods `__init__`, `generate_cards`, `roll_dice`, and `assign_cards`. The `__init__` method initializes the number of players, cards, and unused players and cards. The `generate_cards` method generates a list of cards based on the number of players. The `roll_dice` method returns a random integer between 0 and n-1. The `assign_cards` method assigns cards to players based on the result of the roll.

```
28 class CasinoAgent:
29     def __init__(self, num_players):
30         self.players = [Player(i + 1) for i in range(num_players)]
31         self.cards = self.generate_cards(num_players)
32         self.unused_players = set(range(num_players))
33         self.unused_cards = set(range(num_players))
34
35     def generate_cards(self, n):
36         suits = ["Spades", "Hearts", "Diamonds", "Clubs"]
37         cards = []
38         for i in range(n):
39             value = random.randint(1, 13)
40             suit = random.choice(suits)
41             cards.append(Card(value, suit))
42         return cards
43
44     def roll_dice(self, n):
45         return random.randint(0, n - 1)
46
47     def assign_cards(self):
48         print('\n--- Assigning Cards to Players ---')
49         while self.unused_players and self.unused_cards:
50             player_index = self.roll_dice(len(self.players))
51             card_index = self.roll_dice(len(self.cards))
52             if player_index in self.unused_players and card_index in self.unused_cards:
53                 player = self.players[player_index]
54                 card = self.cards[card_index]
55                 player.assign_card(card)
```

The screenshot shows a code editor with a project named 'PythonProject'. The file explorer on the left shows a directory structure with files like 'Lab9 q1.py', 'Lab9 q2.py', 'task1.py', 'task2.py', 'task3.py', 'task4.py', 'task5.py', 'ab7 q2.py', 'lab7 q5.py', 'lab7 q3.py', 'lab7 q4.py', 'lab 7 q1.py', 'lab 7 q6.py', 'lab 8 q1.py', 'lab 8 q2.py', 'lab 8 q3.py', 'Lab9 q2.py', 'task1.py', and 'task2.py'. The main editor displays the code for 'Lab9 q1.py'.

```
28 class CasinoAgent:
47     def assign_cards(self):
54         card = self.cards[card_index]
55         player.assign_card(card)
56         print(f'{player} receives {card}')
57         self.unused_players.remove(player_index)
58         self.unused_cards.remove(card_index)
59
60     def find_winner(self):
61         print('\n--- Announcing Winner ---')
62         winner = None
63         for player in self.players:
64             if player.card:
65                 if not winner or player.card.card_strength() > winner.card.card_strength():
66                     winner = player
67         if winner:
68             print(f'The winner is {winner} with {winner.card}!')
69         else:
70             print('No winner determined.')
71
72 def main():
73     print(' Welcome to the Casino Game!')
74     n = int(input('Enter the number of players: '))
75     agent = CasinoAgent(n)
76     agent.assign_cards()
77     agent.find_winner()
78     main()
```

The screenshot shows the same code editor as above, but with the 'Run' button clicked. The output window on the right displays the execution results.

```
C:\Users\dell\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\dell\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\dell\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\dell\PycharmProjects\PythonProject\.venv\Scripts\python.exe
Welcome to the Casino Game!
Enter the number of players: 4

--- Assigning Cards to Players ---
Player 3 receives 11 of Hearts
Player 2 receives 8 of Clubs
Player 4 receives 9 of Clubs
Player 1 receives 8 of Spades

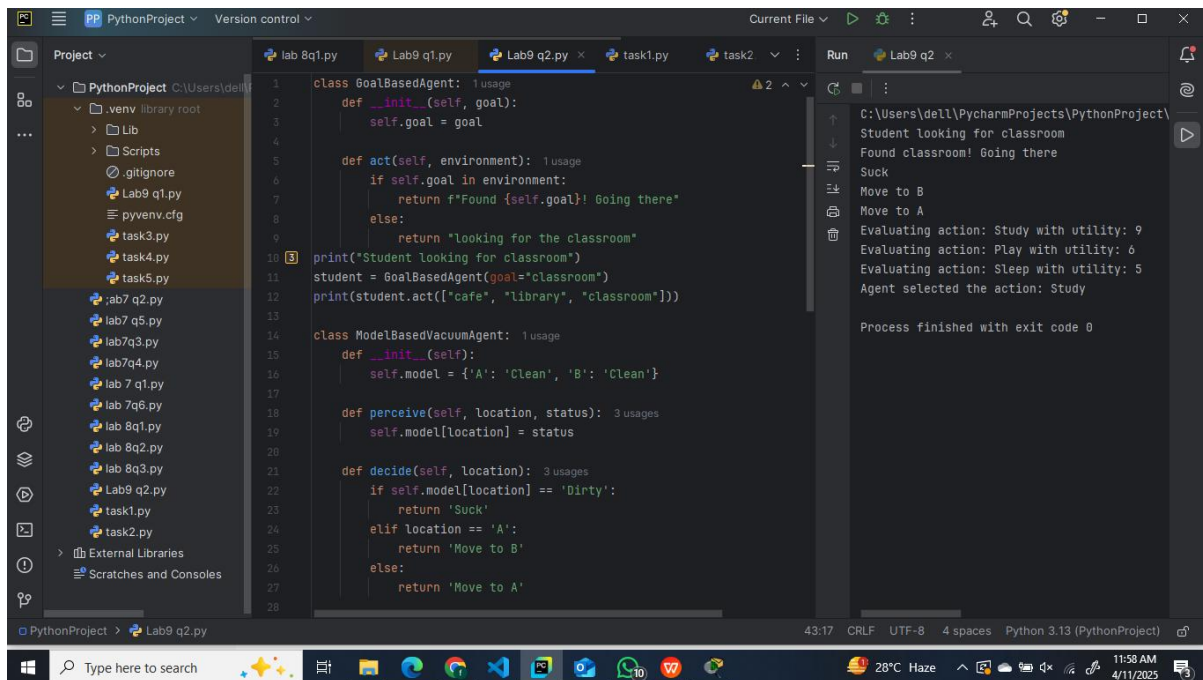
--- Announcing Winner ---
The winner is Player 3 with 11 of Hearts!

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.

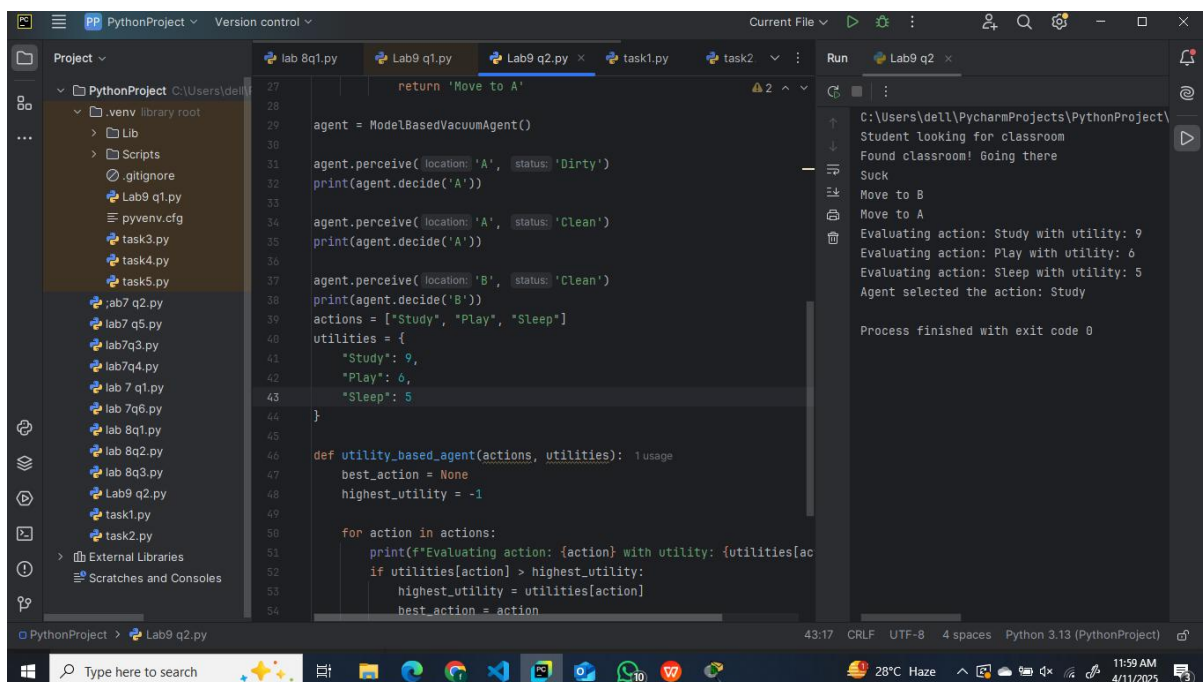
- The program includes the implementation of goal-based agents.
- The program includes the implementation of the model-based agent.
- The program includes the implementation of a utility-based agent.



```
1 class GoalBasedAgent: 1 usage
2     def __init__(self, goal):
3         self.goal = goal
4
5     def act(self, environment): 1 usage
6         if self.goal in environment:
7             return f'Found {self.goal}! Going there'
8         else:
9             return "looking for the classroom"
10
11 print("Student looking for classroom")
12 student = GoalBasedAgent(goal="classroom")
13 print(student.act(["cafe", "library", "classroom"]))
14
15 class ModelBasedVacuumAgent: 1 usage
16     def __init__(self):
17         self.model = {'A': 'Clean', 'B': 'Clean'}
18
19     def perceive(self, location, status): 3 usages
20         self.model[location] = status
21
22     def decide(self, location): 3 usages
23         if self.model[location] == 'Dirty':
24             return 'Suck'
25         elif location == 'A':
26             return 'Move to B'
27         else:
28             return 'Move to A'
```

Run output:

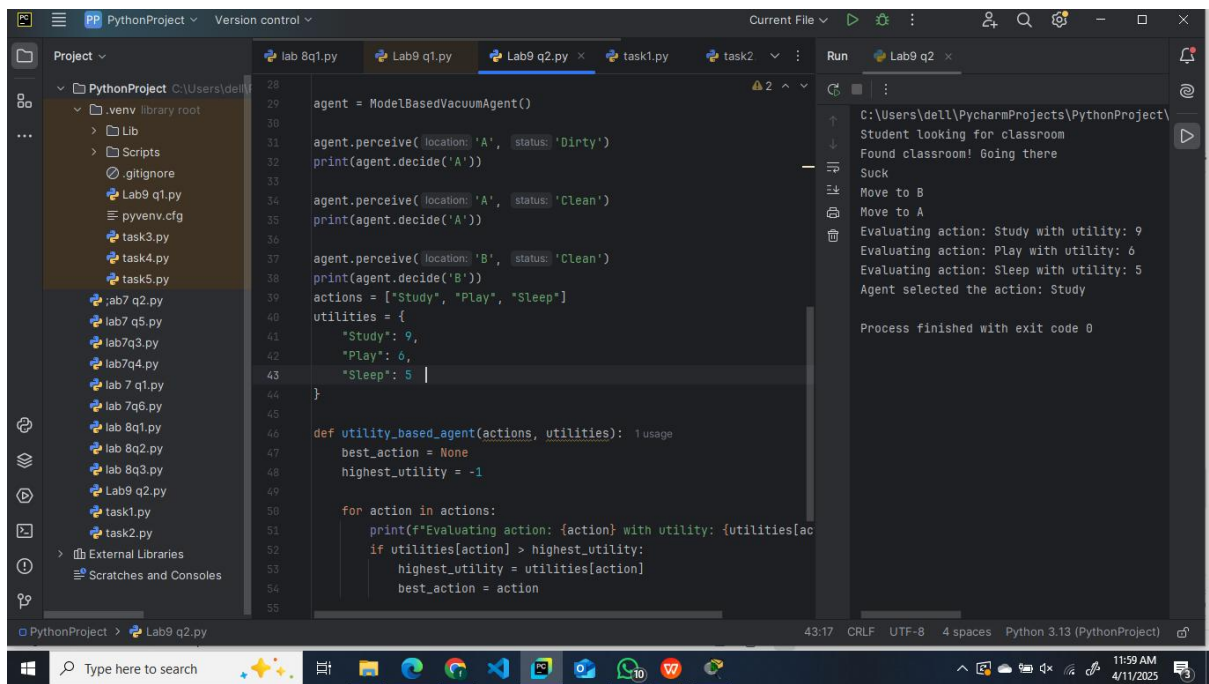
```
Student looking for classroom
Found classroom! Going there
Suck
Move to B
Move to A
Evaluating action: Study with utility: 9
Evaluating action: Play with utility: 6
Evaluating action: Sleep with utility: 5
Agent selected the action: Study
Process finished with exit code 0
```



```
27         return 'Move to A'
28
29 agent = ModelBasedVacuumAgent()
30 agent.perceive(location='A', status='Dirty')
31 print(agent.decide('A'))
32
33 agent.perceive(location='A', status='Clean')
34 print(agent.decide('A'))
35
36 agent.perceive(location='B', status='Clean')
37 print(agent.decide('B'))
38 actions = ["Study", "Play", "Sleep"]
39 utilities = {
40     "Study": 9,
41     "Play": 6,
42     "Sleep": 5
43 }
44
45 def utility_based_agent(actions, utilities): 1 usage
46     best_action = None
47     highest_utility = -1
48
49     for action in actions:
50         print(f"Evaluating action: {action} with utility: {utilities[action]}")
51         if utilities[action] > highest_utility:
52             highest_utility = utilities[action]
53             best_action = action
```

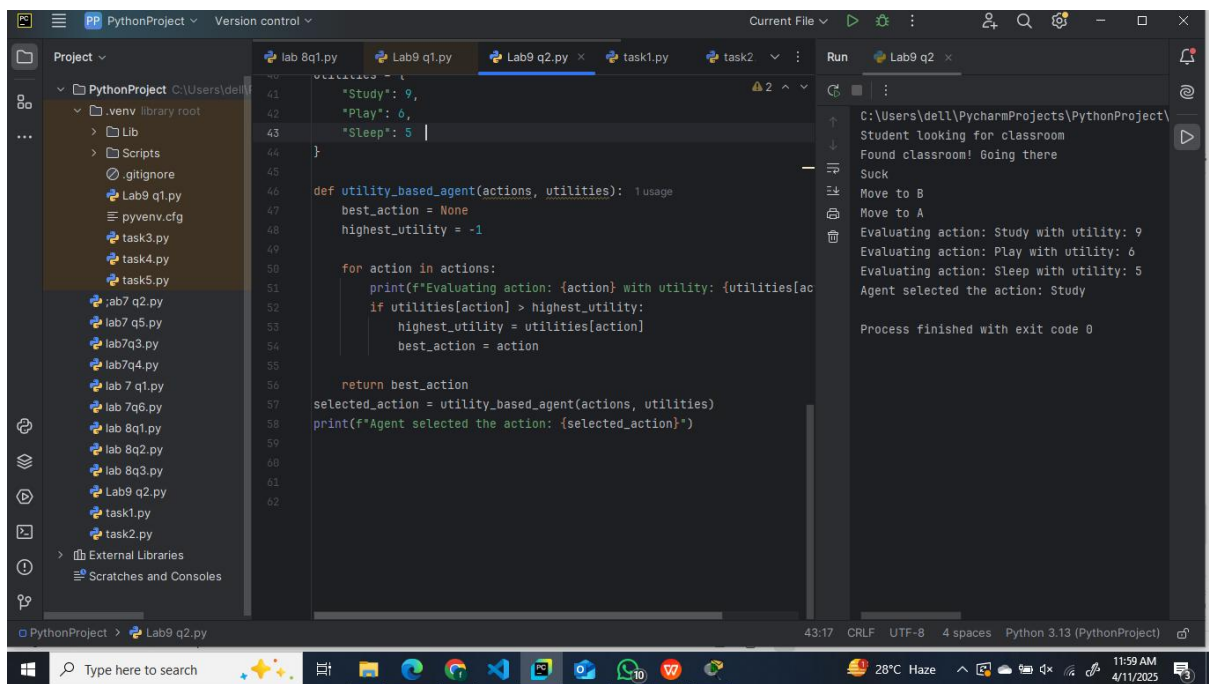
Run output:

```
Student looking for classroom
Found classroom! Going there
Suck
Move to B
Move to A
Evaluating action: Study with utility: 9
Evaluating action: Play with utility: 6
Evaluating action: Sleep with utility: 5
Agent selected the action: Study
Process finished with exit code 0
```

```
28 agent = ModelBasedVacuumAgent()
29
30 agent.perceive((location: 'A', (status: 'Dirty'))
31 print(agent.decide('A'))
32
33 agent.perceive((location: 'A', (status: 'Clean'))
34 print(agent.decide('A'))
35
36
37 agent.perceive((location: 'B', (status: 'Clean'))
38 print(agent.decide('B'))
39 actions = ["Study", "Play", "Sleep"]
40 utilities = {
41     "Study": 9,
42     "Play": 6,
43     "Sleep": 5
44 }
45
46 def utility_based_agent(actions, utilities): 1 usage
47     best_action = None
48     highest_utility = -1
49
50     for action in actions:
51         print(f"Evaluating action: {action} with utility: {utilities[action]}")
52         if utilities[action] > highest_utility:
53             highest_utility = utilities[action]
54             best_action = action
55
56
57 selected_action = utility_based_agent(actions, utilities)
58 print(f"Agent selected the action: {selected_action}")
```

Process finished with exit code 0



```
41     "Study": 9,
42     "Play": 6,
43     "Sleep": 5
44 }
45
46 def utility_based_agent(actions, utilities): 1 usage
47     best_action = None
48     highest_utility = -1
49
50     for action in actions:
51         print(f"Evaluating action: {action} with utility: {utilities[action]}")
52         if utilities[action] > highest_utility:
53             highest_utility = utilities[action]
54             best_action = action
55
56     return best_action
57
58 selected_action = utility_based_agent(actions, utilities)
59 print(f"Agent selected the action: {selected_action}")
```

Process finished with exit code 0