**Bisma Farhat**

**39967**

TASK 1:

import random

class PlayingCard:

    def \_\_init\_\_(self, rank, suit):

        self.rank = rank

        self.suit = suit

    def \_\_str\_\_(self):

        return f"{self.rank} of {self.suit}"

    def get\_priority(self):

        suit\_priority = {'Spades': 4, 'Hearts': 3, 'Diamonds': 2, 'Clubs': 1}

        return self.rank + suit\_priority[self.suit] \* 10

class Gambler:

    def \_\_init\_\_(self, name):

        self.name = name

        self.card = None

    def assign\_card(self, card):

        self.card = card

    def \_\_str\_\_(self):

        return f"{self.name} with card {self.card}"

class CasinoGame:

    def \_\_init\_\_(self, player\_count):

        self.gamblers = [Gambler(f"Gambler {i + 1}") for i in range(player\_count)]

        self.deck = self.create\_deck()

        self.assigned\_cards = set()

    def create\_deck(self):

        suits = ['Spades', 'Hearts', 'Diamonds', 'Clubs']

        return [PlayingCard(rank, suit) for suit in suits for rank in range(1, 14)]

    def roll\_dice(self):

        return random.randint(1, len(self.gamblers)), random.randint(1, len(self.deck))

    def assign\_cards(self):

        for \_ in range(len(self.gamblers)):

            while True:

                player\_roll, card\_roll = self.roll\_dice()

                if player\_roll - 1 not in self.assigned\_cards and card\_roll - 1 not in self.assigned\_cards:

                    gambler = self.gamblers[player\_roll - 1]

                    card = self.deck[card\_roll - 1]

                    gambler.assign\_card(card)

                    self.assigned\_cards.add(player\_roll - 1)

                    self.assigned\_cards.add(card\_roll - 1)

                    break

    def determine\_winner(self):

        winner = max(self.gamblers, key=lambda gambler: gambler.card.get\_priority())

        return winner

    def play\_game(self):

        print("Starting the game...\n")

        self.assign\_cards()

        for gambler in self.gamblers:

            print(gambler)

        winner = self.determine\_winner()

        print(f"\nThe winner is: {winner.name} with card {winner.card}!")

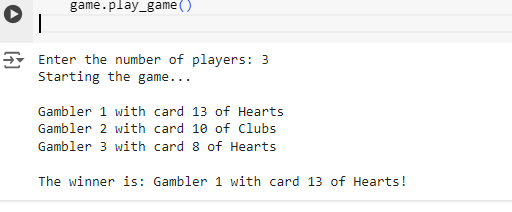
if \_\_name\_\_ == "\_\_main\_\_":

    player\_count = int(input("Enter the number of players: "))

    game = CasinoGame(player\_count)

    game.play\_game()

Output:



TASK 2:

import random

class GoalBasedAgent:

    def \_\_init\_\_(self, initial\_state, goal\_state):

        self.state = initial\_state

        self.goal\_state = goal\_state

    def act(self):

        if self.state < self.goal\_state:

            self.state += 1

            print(f"[Goal-Based Agent] Moved to state {self.state} to reach goal {self.goal\_state}.")

        elif self.state > self.goal\_state:

            self.state -= 1

            print(f"[Goal-Based Agent] Moved to state {self.state} to reach goal {self.goal\_state}.")

        else:

            print("[Goal-Based Agent] Goal achieved!")

class ModelBasedAgent:

    def \_\_init\_\_(self, initial\_state):

        self.state = initial\_state

        self.model = {}

    def update\_model(self, action):

        self.model[self.state] = action

    def act(self):

        action = "Increase" if self.state < 10 else "Stop"

        if action == "Increase":

            self.state += 1

        self.update\_model(action)

        print(f"[Model-Based Agent] State: {self.state}, Action: {action}, Model: {self.model}")

class UtilityBasedAgent:

    def \_\_init\_\_(self, initial\_state):

        self.state = initial\_state

    def utility\_function(self):

        return -abs(10 - self.state)

    def act(self):

        if self.state < 10:

            self.state += 1

        elif self.state > 10:

            self.state -= 1

        print(f"[Utility-Based Agent] Current State: {self.state}, Utility: {self.utility\_function()}")

def goal\_based\_scenario():

    print("----- Goal-Based Agent Scenario -----")

    goal\_agent = GoalBasedAgent(initial\_state=0, goal\_state=5)

    while goal\_agent.state != goal\_agent.goal\_state:

        goal\_agent.act()

def model\_based\_scenario():

    print("\n----- Model-Based Agent Scenario -----")

    model\_agent = ModelBasedAgent(initial\_state=random.randint(0, 10))

    for \_ in range(12):

        model\_agent.act()

def utility\_based\_scenario():

    print("\n----- Utility-Based Agent Scenario -----")

    utility\_agent = UtilityBasedAgent(initial\_state=0)

    for \_ in range(12):

        utility\_agent.act()

def main():

    goal\_based\_scenario()

    model\_based\_scenario()

    utility\_based\_scenario()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Output:

