RIPHAH
INTERNATIONAL
UNIVERSITY

Artificial Intelligence Lab

Name: Maryam Malik

Sap ID: 45369

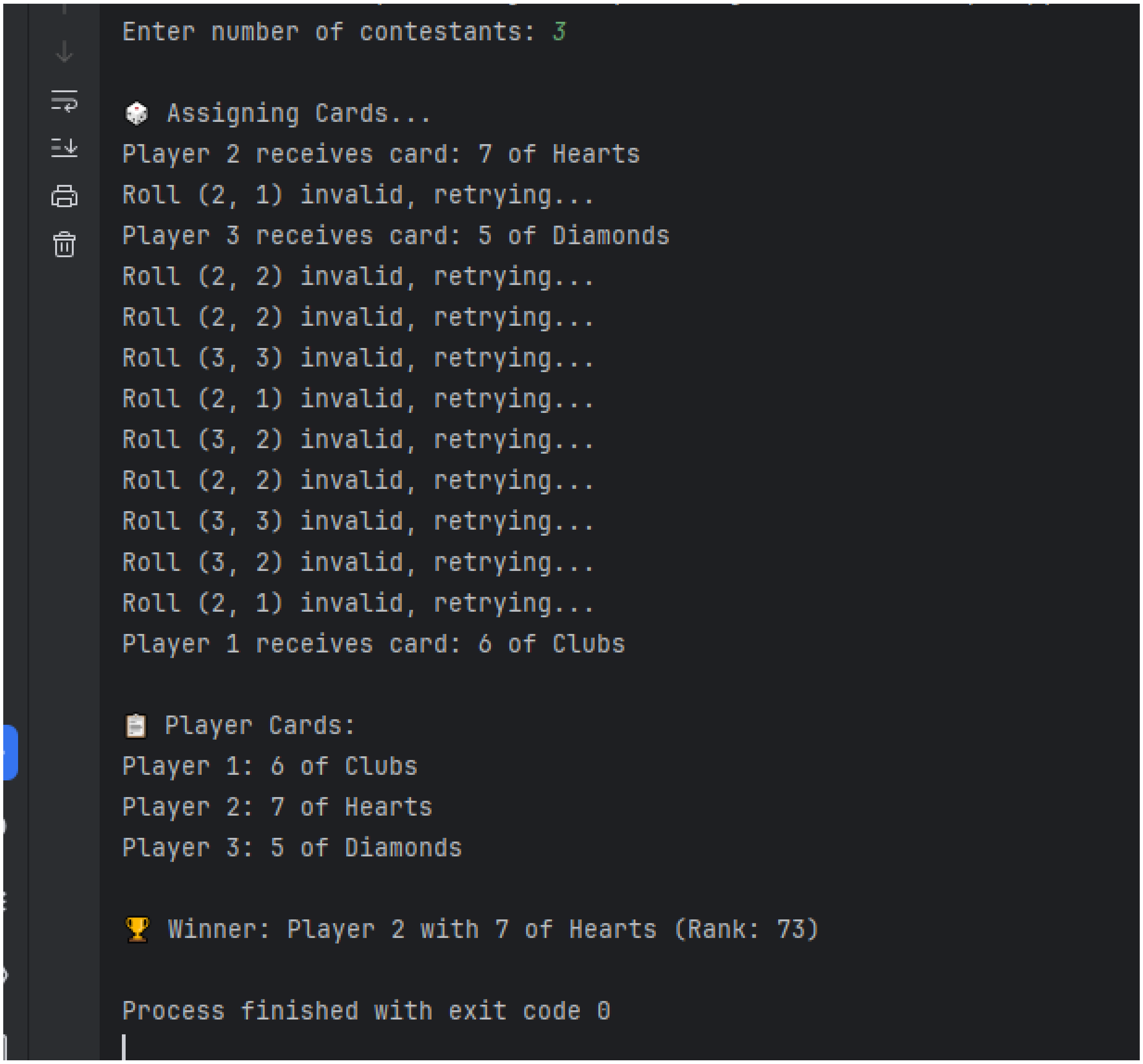
Batch: BSCS-6th semester

Instructor: Ayesh Akram

Task 01

```
import random

# Suits Priority (Higher number = higher priority)
suit_priority = {
    "Spades": 4,
    "Hearts": 3,
    "Diamonds": 2,
    "Clubs": 1
}

# Card Class
class Card:
    def __init__(self, value, suit):
        self.value = value
        self.suit = suit
        self.valid = True

    def __str__(self):
        face_cards = {11: "Jack", 12: "Queen", 13: "King", 14: "Ace"}
        val_str = face_cards.get(self.value, str(self.value))
        return f"{val_str} of {self.suit}"

    def rank(self):
        return self.value * 10 + suit_priority[self.suit]  # Higher total = better card

# Player Class
class Player:
    def __init__(self, id):
        self.id = id
        self.valid = True
        self.card = None
    def __str__(self):
        return f"Player {self.id}"

# Casino Agent Class
class CasinoAgent:
    def __init__(self, n):
        self.n = n
        self.players = [Player(i + 1) for i in range(n)]
        self.cards = self.generate_cards(n)

    def generate_cards(self, n):
        suits = list(suit_priority.keys())
        cards = []
        for _ in range(n):
            value = random.randint(2, 14)  # 2 to Ace (14)
            suit = random.choice(suits)
            cards.append(Card(value, suit))
        return cards

    def roll_dice(self, sides=None):
        return random.randint(1, sides or self.n)

    def assign_cards(self):
        print("\n Assigning Cards...")
        assigned = 0
        while assigned < self.n:
            p_roll = self.roll_dice()
            c_roll = self.roll_dice()
            player = self.players[p_roll - 1]
            card = self.cards[c_roll - 1]
            if player.valid and card.valid:
                player.card = card
                player.valid = False
                card.valid = False
                print(f"{player} receives card: {card}")
                assigned += 1
            else:
                print(f"Roll ({p_roll}, {c_roll}) invalid, retrying...")

    def display_all_cards(self):
        print("\n Player Cards:")
        for p in self.players:
            print(f"{p}: {p.card}")

    def declare_winner(self):
        valid_players = [p for p in self.players if p.card]
        winner = max(valid_players, key=lambda p: p.card.rank())
        print(f"\n Winner: {winner} with {winner.card} (Rank: {winner.card.rank()})")

# MAIN FUNCTION
def main():
    try:
        n = int(input("Enter number of contestants: "))
        if n < 1:
            raise ValueError("Number of contestants must be at least 1.")
    except ValueError as e:
        print(f"Invalid input: {e}")
        return

    agent = CasinoAgent(n)
    agent.assign_cards()
    agent.display_all_cards()
    agent.declare_winner()

# Run the game
if __name__ == "__main__":
    main()
```

Output

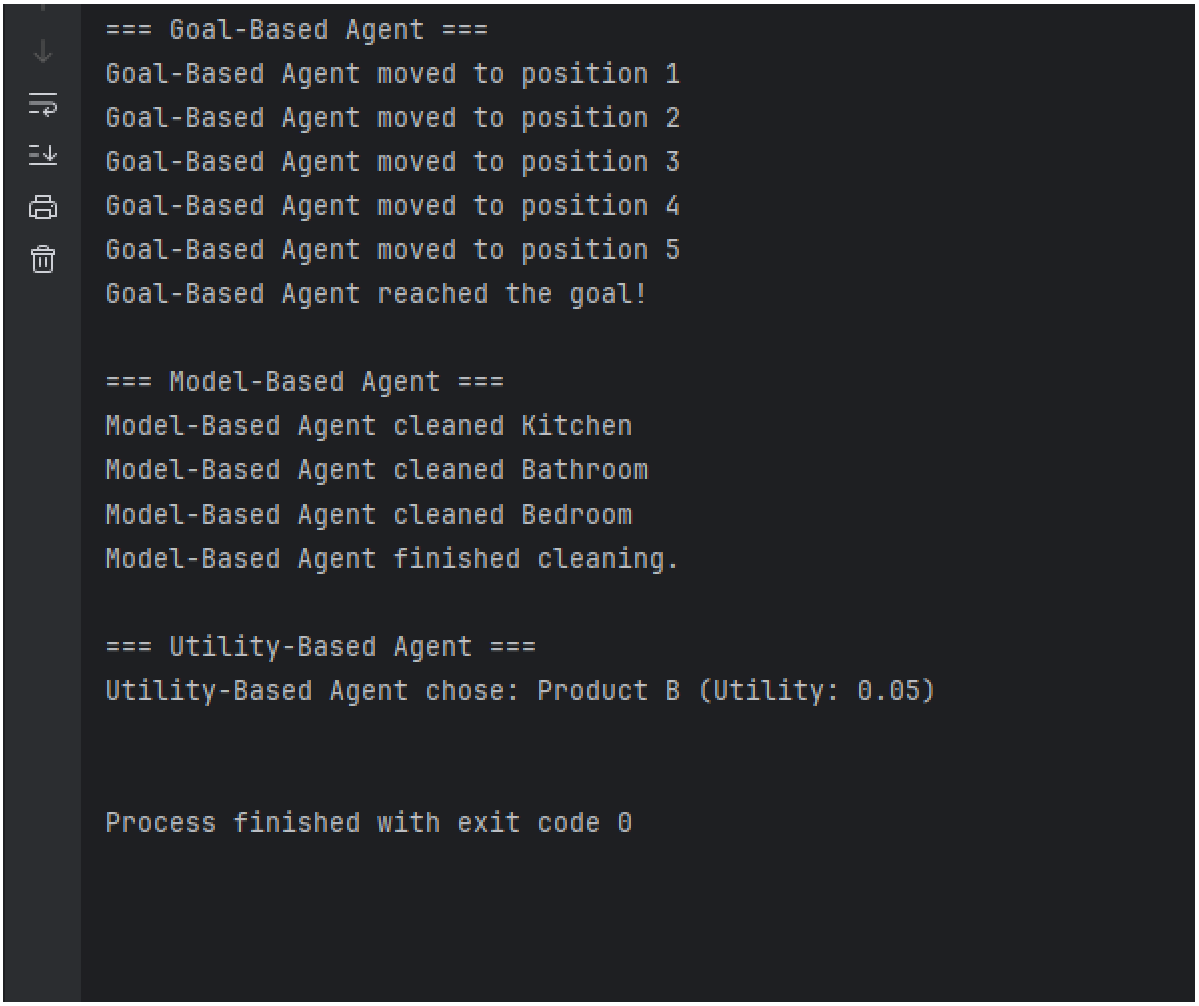


```
Enter number of contestants: 3

🎲 Assigning Cards...
Player 2 receives card: 7 of Hearts
Roll (2, 1) invalid, retrying...
Player 3 receives card: 5 of Diamonds
Roll (2, 2) invalid, retrying...
Roll (2, 2) invalid, retrying...
Roll (3, 3) invalid, retrying...
Roll (2, 1) invalid, retrying...
Roll (3, 2) invalid, retrying...
Roll (2, 2) invalid, retrying...
Roll (3, 3) invalid, retrying...
Roll (3, 2) invalid, retrying...
Roll (2, 1) invalid, retrying...
Player 1 receives card: 6 of Clubs

📋 Player Cards:
Player 1: 6 of Clubs
Player 2: 7 of Hearts
Player 3: 5 of Diamonds

🏆 Winner: Player 2 with 7 of Hearts (Rank: 73)

Process finished with exit code 0
```

Task 02

```
import random


# ------------------------
# 1. Goal-Based Agent
# ------------------------
class GoalBasedAgent:
    def __init__(self, goal_position):
        self.position = 0
        self.goal = goal_position

    def move(self):
        while self.position != self.goal:
            if self.position < self.goal:
                self.position += 1
            else:
                self.position -= 1
            print(f"Goal-Based Agent moved to position {self.position}")
        print("Goal-Based Agent reached the goal!\n")



# ------------------------
# 2. Model-Based Agent
# ------------------------
class ModelBasedAgent:
    def __init__(self, rooms):
        self.rooms = {room: 'dirty' for room in rooms}
        self.current_room = random.choice(rooms)

    def perceive(self):
        return self.rooms[self.current_room]

    def update_model(self, room, status):
        self.rooms[room] = status

    def act(self):
        for room in self.rooms:
            self.current_room = room
            status = self.perceive()
            if status == 'dirty':
                print(f"Model-Based Agent cleaned {room}")
                self.update_model(room, 'clean')
            else:
                print(f"Model-Based Agent skipped {room} (already clean)")
        print("Model-Based Agent finished cleaning.\n")


# -------------------------
# 3. Utility-Based Agent
# -------------------------
class UtilityBasedAgent:
    def __init__(self, products):
        self.products = products  # list of dicts with 'name', 'price', and 'rating'

    def calculate_utility(self, product):
        # Higher rating and lower price = better utility
        return product['rating'] / product['price']

    def choose_best_product(self):
        best_product = max(self.products, key=self.calculate_utility)
        print(f"Utility-Based Agent chose: {best_product['name']} (Utility:
{self.calculate_utility(best_product):.2f})\n")


# -------------------------
# Main Program to Run All Agents
# -------------------------
if __name__ == "__main__":
    print("=== Goal-Based Agent ===")
    goal_agent = GoalBasedAgent(goal_position=5)
    goal_agent.move()

    print("=== Model-Based Agent ===")
    model_agent = ModelBasedAgent(rooms=['Kitchen', 'Bathroom', 'Bedroom'])
    model_agent.act()

    print("=== Utility-Based Agent ===")
    products = [
        {'name': 'Product A', 'price': 100, 'rating': 4.5},
        {'name': 'Product B', 'price': 80, 'rating': 4.0},
        {'name': 'Product C', 'price': 120, 'rating': 5.0},
    ]
    utility_agent = UtilityBasedAgent(products)
    utility_agent.choose_best_product()
```

Output



```
=== Goal-Based Agent ===
Goal-Based Agent moved to position 1
Goal-Based Agent moved to position 2
Goal-Based Agent moved to position 3
Goal-Based Agent moved to position 4
Goal-Based Agent moved to position 5
Goal-Based Agent reached the goal!

=== Model-Based Agent ===
Model-Based Agent cleaned Kitchen
Model-Based Agent cleaned Bathroom
Model-Based Agent cleaned Bedroom
Model-Based Agent finished cleaning.

=== Utility-Based Agent ===
Utility-Based Agent chose: Product B (Utility: 0.05)


Process finished with exit code 0
```