Vivekanand Sahu vsahu@ucsd.edu

AGENT BASED 21 CARD GAME

Index:

Vivekanand Sahu vsahu@ucsd.edu

I. Game Introduction and Rules:

1.1 Game Rules:

Objective: Achieve the highest total score under or equal to 21. The highest valid score wins. If all exceed 21, no winner is declared.

Players:

User: Plays the game with AI Agents and interacts with other Game assisting agents.
3 AI Agents: LLM models act like players and make smart moves.

Gameplay:

- Just like blackjack every player starts with 2 random cards. (Assumption: If they start with 1 card it becomes obvious that every player will have to take another move to get closer to 21).
- Players can then Hit or Stand to ask for an additional card or pass.
- Cards are randomly generated with values between 2 and 11. (Assumption: Since the problem statement says 2 to 11. 'A' cannot become 1, it is always 11)
- Scores are calculated by summing the scores for individual cards, where cards from 2 to 10 get the score of the corresponding number. Face cards get a score of 10. And A gets a score of 11.

Winning:

- After every round the scores are calculated for each player.
- Similarly, 3 completely rounds are played, and winners of each round is declared.
- Towards the end the player with the highest cumulative points is chosen to become the winner. (Interpretation: Since there are per round winners and a final winner, cumulative sum of points approach to select the game winner was chosen).

Key Points:

- Players get to see the initial 2 cards of other players.
- When players (including the user) ask for additional cards all the other players get to know their response (yes/no only. Not the exact card). Only the system knows the response and cards.
- The final cards are revealed at the end of every round to declare a round's winner.

II. Approach and Framework Selection:

2.1 Approach:

LLMs can simulate realistic players in games by mimicking decision-making and diverse strategies, enhancing immersion. They also act as assistants, offering rule explanations, strategy suggestions, or analyzing moves. With natural language interaction, LLMs create dynamic, engaging experiences, blending AI intelligence with human-like communication to enrich gameplay for users and players.

2.2 Framework:

CrewAI simplifies managing multiple AI agents with diverse behaviors, making it ideal for simulating a "crew" of decision-makers. **LangChain** streamlines integrating LLMs for reasoning, memory, and interaction, enabling dynamic, human-like decision-making. Together, they provide modular, scalable tools for building intelligent, interactive agents in complex tasks like simulating players in games. The **Flow** feature of the **CrewAI** framework was chosen due to its ability to handle sequential tasks, ensuring smooth, ordered execution of actions and decisions throughout the game process.

2.3 Defining Agents, Tasks, Tools, and Crew:

Agents: Agents are autonomous entities in CrewAI designed to perform specific actions or make decisions, often powered by AI models, scripts, or logic frameworks.

Tasks: Tasks are defined objectives or actions assigned to agents, guiding their behavior and decisions to achieve a desired outcome within the CrewAI environment.

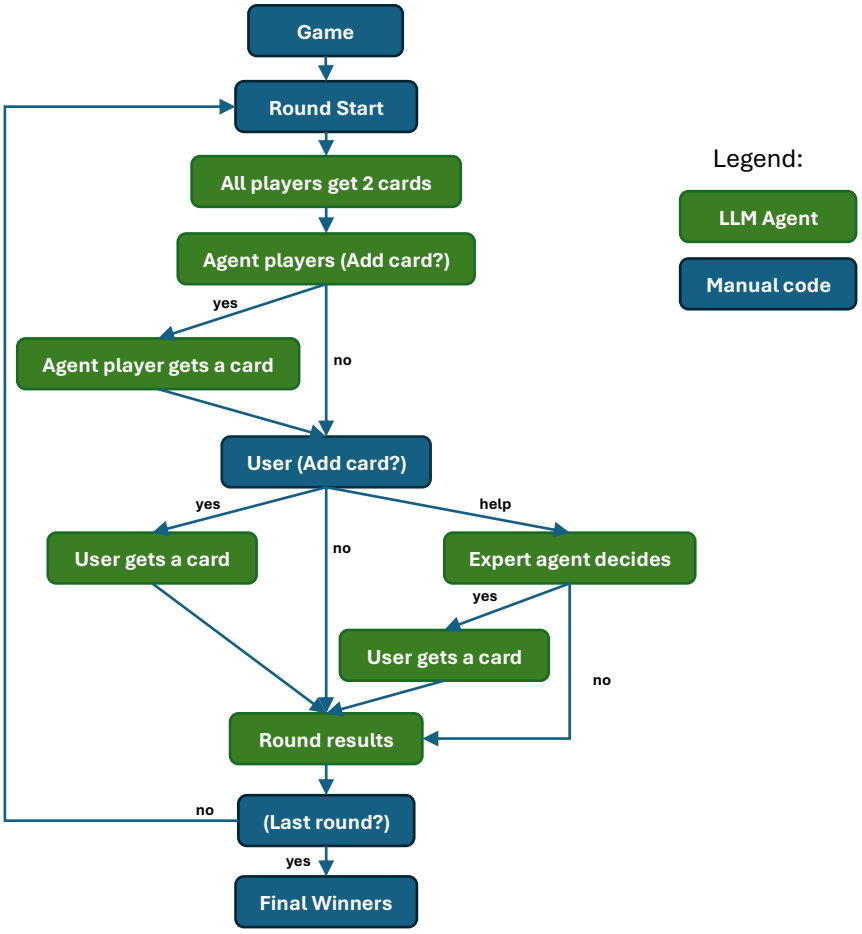
Tools: Tools are functional modules or resources agents can use to complete tasks, such as APIs, data processors, or external systems integrated into CrewAI workflows.

Crew: The Crew is a collection of agents working collaboratively or independently to accomplish tasks, leveraging their individual capabilities and tools to achieve shared goals.

Vivekanand Sahu vsahu@ucsd.edu

III. Game flow and Step:

Flowchart diagram:



3.1 Steps and entities used for each step:

1. Distributing 2 cards to all the Players to start the game.
   Crew: DistributeCrew
   Agent: game_host
   Task: distribute_cards

   Example:

```
# Agent: Expert Game dealer of playing cards.
## Final Answer:
{'Player1': ['5', 'J'], 'Player2': ['A', '7'], 'Player3': ['3', 'K'], 'User': ['Q', '9']}
```

2. Ask every Agentic players if they want to take an additional card and record their response.
   Crew: PlayersCrew
   Agent: players
   Task: add_card_for_player

   Example:

```
# Agent: CrewAI expert cards game player Player1.
## Final Answer:
yes
```

```
# Agent: CrewAI expert cards game player Player2.
## Final Answer:
no
```

```
# Agent: CrewAI expert cards game player Player3.
## Final Answer:
yes
```

3. Add card to the agentic players if they asked for it.
   Crew: AddCardCrew
   Agent: card_adder
   Task: card_adder_for_user

   Example:

```
# Agent: CrewAI expert card adder for the specified player Player1.
## Final Answer:
{'Player1': ['5', 'J', 'K'], 'Player2': ['A', '7'], 'Player3': ['3', 'K'], 'User': ['Q', '9']}
```

```
# Agent: CrewAI expert card adder for the specified player Player3.
## Final Answer:
{'Player1': ['5', 'J', 'K'], 'Player2': ['A', '7'], 'Player3': ['3', 'K', '7'], 'User': ['Q', '9']}
```

4. Ask the user if he/she wants to take an additional card or need some expert advise. Record the response as ‘yes’ or ‘no’ or ‘help’.

   Example:

```
Do you want another card? Write "yes"/"no"/"help"
```

5. Add card to the User if asked for it.
   Crew: AddCardCrew
   Agent: card_adder
   Task: card_adder_for_user

   Example:

```
# Agent: CrewAI expert card adder for the specified player User.
## Final Answer:
{'Player1': ['5', 'J', 'K'], 'Player2': ['A', '7'], 'Player3': ['3', 'K', '7'], 'User': ['Q', '9', '5']}
```

6. If the User asks for help, call the Expert agent if they need help and let the agent decide for the user.
   Crew: PlayersCrew, AddCardCrew
   Agent: players, card_adder
   Task: add_card_for_player, card_adder_for_user

   Example (Not linked with the example):

```
# Agent: CrewAI expert cards game player User.
## Final Answer:
yes
```

```
# Agent: CrewAI expert card adder for the specified player User.
## Final Answer:
{'Player1': ['9', 'K'], 'Player2': ['6', '2'], 'Player3': ['A', 'J'], 'User': ['3', 'Q', '7']}
```

7. Calculate the Scores of all the players and declare round winners
   Crew: WinnerDeciderCrew
   Agent: scores_calculator
   Task: calculate_scores
   Tool: MyCustomTool

   Example:

```
# Agent: Expert card score calculator and winner decider
## Thought: To solve the problem, I need to calculate the total score for each player using the provided card scoring rules. I will use the available to
## Using tool: Card Score Calculator
## Tool Input:
"{\"players\": {\"Player1\": [\"5\", \"J\", \"K\"], \"Player2\": [\"A\", \"7\"], \"Player3\": [\"3\", \"K\", \"7\"], \"User\": [\"Q\", \"9\", \"5\"]}}"
## Tool Output:
{'Player1': 0, 'Player2': 18, 'Player3': 20, 'User': 0}


# Agent: Expert card score calculator and winner decider
## Final Answer:
{'Player1': 0, 'Player2': 18, 'Player3': 20, 'User': 0}
```

```
 {'Player1': 0, 'Player2': 18, 'Player3': 20, 'User': 0}
The winner is Player3 with a score of 20
Cards at the end of this Round {'Player1': ['5', 'J', 'K'], 'Player2': ['A', '7'], 'Player3': ['3', 'K', '7'], 'User': ['Q', '9', '5']}
```

8. Repeat steps 1 to 7 for round 2 and round 3. The final winner/winners win the game on the basis of their cumulative scores of each round.

   Example:

```
------------------------------ Final Results -------------------------------------

 {'Player1': 19, 'Player2': 44, 'Player3': 41, 'User': 38}
The winner is Player2 with a score of 44
Game Over
```

3.2 Agents and Task instructions:

1. Agent and task used to simulate card dealer that distributes 2 cards to each player:

Agent: game_host

```yaml
game_host:
  role: >
    Expert Game dealer of playing cards.
  goal: >
    Randomly generate 2 cards from a deck of playing cards for all the players represented as keys of the dictionary {player_cards}.
    The cards should be represented by numbers as '2' or faces as 'K' and ace as 'A'.
    Update the list of cards for all the players in the dictionary with the corresponding cards.
    Finally return the updated dictionary. The output should contain only the updated dictionary and no additional text.
  backstory: >
    You have decades of expereince in distributing cards to players.
```

Task: distribute_cards

```yaml
distribute_cards:
  description: >
    Randomly generate 2 cards from a deck for all the players represented as keys of the dictionary {player_cards}.
  expected_output: >
    An updated dictionary containing the list of cards for all the players in the dictionary.
    The cards should be represented by numbers as '2' or faces as 'K' and ace as 'A'.
    The output should contain only the updated dictionary and no additional text.
  agent: game_host
```

2. Agent and task used to simulate Agentic players:

Agent: players

```yaml
players:
  role: >
    CrewAI expert cards game player {player}.
  goal: >
    Compete with other players represented as keys of the dictionary {player_cards} and their values represent the list of cards they have.
    Calculate the total score of all the players using their cards where numbers 2 to 10 have their corresponding number as scores, face
    cards are scored 10 and A is scored 11.
    Decide whether or not to ask for an additional card that will make the total score of the player {player} greater than other player's
    total score but does not exceed 21.
    The output should only be 'yes' if {player} should take another card or 'no' if player should not take another card.
  backstory: >
    You have decades of expereince in computing the total score of cards for every player and determining whether or not to take another
    card such that your total score becomes greater than other player's total score but does not exceed 21.
```

Task: add_card_for_player

```yaml
add_card_for_player:
  description: >
    Compete with other players represented as keys of the dictionary {player_cards} and their values represent the list of cards they have.
    Calculate the total score of all the players using their cards where numbers 2 to 10 have their corresponding number as scores,
    face cards are scored 10 and A is scored 11.
    Decide whether or not to ask for an additional card that will make the total score of the player {player} greater than other player's
    total score but does not exceed 21.
  expected_output: >
    The output should only be 'yes' if {player} should take another card or 'no' if player should not take another card.
  agent: players
```

3. Agent and task used to simulate card adder dealer that gives an additional card when asked for it:

Agent: card_adder

```
card_adder:
  role: >
    CrewAI expert card adder for the specified player {player}.
  goal: >
    Randomly generate 1 card from a deck of playing cards for the player {player} represented as a keys of the dictionary of
    players {player_cards}.
    The card should be represented by numbers as '2' or faces as 'K' and ace as 'A'.
    Update the list of cards for the player {player} in the dictionary with the additional card.
    Finally return the updated dictionary. The output should contain only the updated dictionary and no additional text.
  backstory: >
    You have decades of expereince in adding a random card to the list of cards of the specified player.
```

Task: card_adder_for_user

```
card_adder_for_user:
  description: >
    Randomly generate 1 card from a deck of playing cards for the player {player} represented as a keys of the dictionary of
    players {player_cards}.
    The card should be represented by numbers as '2' or faces as 'K' and ace as 'A'.
    Update the list of cards for the player {player} in the dictionary with the additional card.
  expected_output: >
    Finally return the updated dictionary. The output should contain only the updated dictionary and no additional text.
  agent: card_adder
```

4. Agent and task used to simulate an expert who can be reached for help by the User:

Task: add_card_for_player

```
calculate_scores:
  description: >
    Calculate the total score of all the players represented as keys of the dictionary {player_cards} and their values represent
    the list of cards they have.
    Use the cards scoring metric where numbers 2 to 10 have their corresponding number as scores, face cards are scored 10 and A
    is scored 11.
    If the total score of any player if more than 21 then the score of this player becomes 0.
  expected_output: >
    Else return only a dictionary like {player_cards} where the values should be the total scores of the corresponding players.
  agent: scores_calculator
```

Agent: scores_calculator

```
scores_calculator:
  role: >
    Expert card score calculator and winner decider
  goal: >
    Calculate the total score of all the players represented as keys of the dictionary {player_cards} and their values represent the
    list of cards they have.
    Use the cards scoring metric where numbers 2 to 10 have their corresponding number as scores, face cards are scored 10 and A is
    scored 11.
    If the total score of any player if more than 21 then the score of this player becomes 0.
    Finally return only a dictionary like {player_cards} where the values should be the total scores of the corresponding players.
  backstory: >
    You have decades of expereince in computing the total score of cards for every player.
```

3.3 Main Flow functioning:

The CrewAI Flow functionality enables us to define a flow of execution. This has been used to put each task in a function and arrange its order depending upon occurrence in the game.

- Main Class: This class is run to start the game.

```
class GameFlow(Flow[GameState]):
```

- Round starter function: This function is executed when the game begins.

```
@start()
def dealer_distributes_cards(self):
```

- Following steps: These functions wait for the completion of some previous function before their execution begins.

```
@listen(dealer_distributes_cards)
def agent_player_decides(self):
```

```
@listen(agent_player_decides)
def user_decides_and_adds(self):
```

```
@listen(and_(dealer_distributes_cards, agent_player_decides, user_decides_and_adds))
def round_winner_announcement(self):
```

IV. Implementation instructions:

4.1 Steps to run this game:

Step 1. Install CrewAI library

```
!pip install crewai
```

Step 2: Create a Flow for the game

```
!crewai create flow card_game
```

Step 3: Install all the packages at the right location

```
%cd /content/card_game
```

```
!crewai install
```

Step 4: Make sure to include all the code from the attached zip file at the desired folder.

Include the following folders in the crews folder :

1. add_card_crew
2. distribute_crew
3. players_crew
4. winner_decider_crew

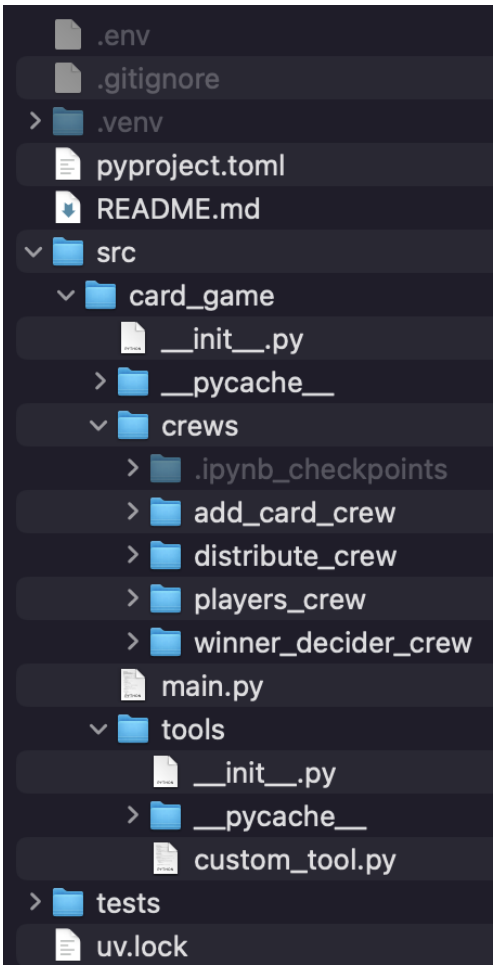
Update the following files using the files in the attached zip file:

1. main.py
2. custom_tool.py

Update the ChatGPT Api key in this file:

1. .env

The final folder structure should look like this:



Step 5: Run this command to start playing the game:

```
!crewai flow kickoff
```

V. Future Scope:

5.1 Future enhancements

- **Optimizing Response Time:** To reduce response time, leveraging lightweight or on-device open-source LLMs, or employing quantized models, can significantly improve efficiency and speed. This approach minimizes latency while maintaining a high level of performance, ensuring a smoother user experience.

- **Improving Response Accuracy:** While accuracy was initially handled with assert statements, it can be further enhanced by implementing continuous feedback loops. These loops would provide real-time input to the LLM, refining its responses over time and ensuring better decision-making. Additionally, fine-tuning the model on domain-specific data can enhance its performance.

5.2 Attached notebook for reference

Please refer the attached notebook called **CrewAI_flow-game.ipynb** – for an example.

Points to remember:

1. When evaluating the system's responses, focus only on the white text that is visible to the user. This is the content the player interacts with during the game.
2. Ignore any colored outputs, as these are internal agent responses that are not shown to the user in the GUI.
3. Additionally, disregard the warning message about "Overriding of current TracerProvider is not allowed," as it does not affect the gameplay experience.