Teaching ChatGPT the "Diamonds" Card Game

Sahana Balajee

1 Problem Statement

The aim of the experiment is to help ChatGPT understand the rules of the "Diamonds" card game and then guide it to write code for a playable version

2 Methodology

I dove into teaching ChatGPT the "Diamonds" card game, explaining the rules step-by-step. To check its understanding, I had it explain the steps back to me. We then brainstormed strategies: keeping track of opponent's cards, and using the revealed diamond as a guide for bidding. However, to avoid giving away points, I suggested bidding low with 2s and 3s for high bids, while saving my actual high cards for later rounds. Confident in its grasp, I took out the spades and challenged ChatGPT to some practice games. This playtime proved fruitful as together we discovered a new strategy: always bid one higher than the revealed diamond, except for aces where we bid the lowest ranked card.
Upon this discovery, I tasked ChatGPT with writing code based on our strategies. We encountered some errors initially, but I asked ChatGPT to fix them. Wanting to push the boundaries, I requested code that factored in the opponent's cards as well. Through this iterative process of explanation, playing, strategising, and coding, I successfully guided ChatGPT towards a deep understanding of Diamonds and its transformation into a playable game.

3 Learnings

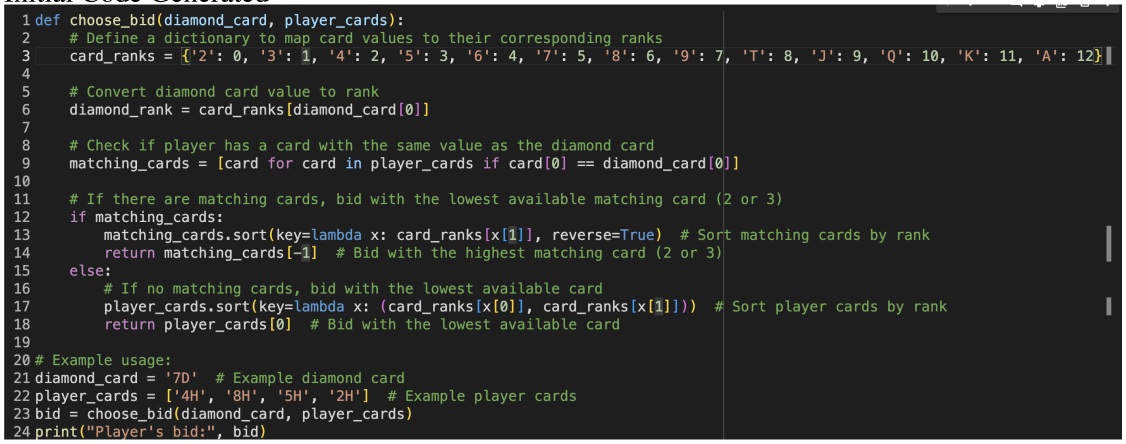
Guiding ChatGPT through the "Diamonds" card game proved to be a valuable learning experience for both. While it required clear explanations initially, ChatGPT ultimately grasped the game's complexities. Our collaboration led to the discovery of a new, potentially winning strategy. However, while ChatGPT excelled at generating code rapidly, initial errors arose, highlighting the need for human oversight. Interestingly, teaching the game not only benefitted ChatGPT's understanding but also deepened my own knowledge through explanation and addressing its questions. The experience underscored the power of human-AI collaboration – combining human expertise with AI's learning abilities can unlock greater potential.

4 Reflections

Teaching ChatGPT "Diamonds" became a journey of mutual learning. While initial explanations ensured its grasp of the game, the process solidified my own knowledge as well. Our collaboration led to a new, potentially winning strategy, highlighting the power of combining human intuition with AI's analytical abilities. Code generation showcased AI's speed, but errors revealed the need for human oversight. This experience underscored the potential of human-AI partnerships, where expertise and creativity can be amplified by AI's learning capabilities.

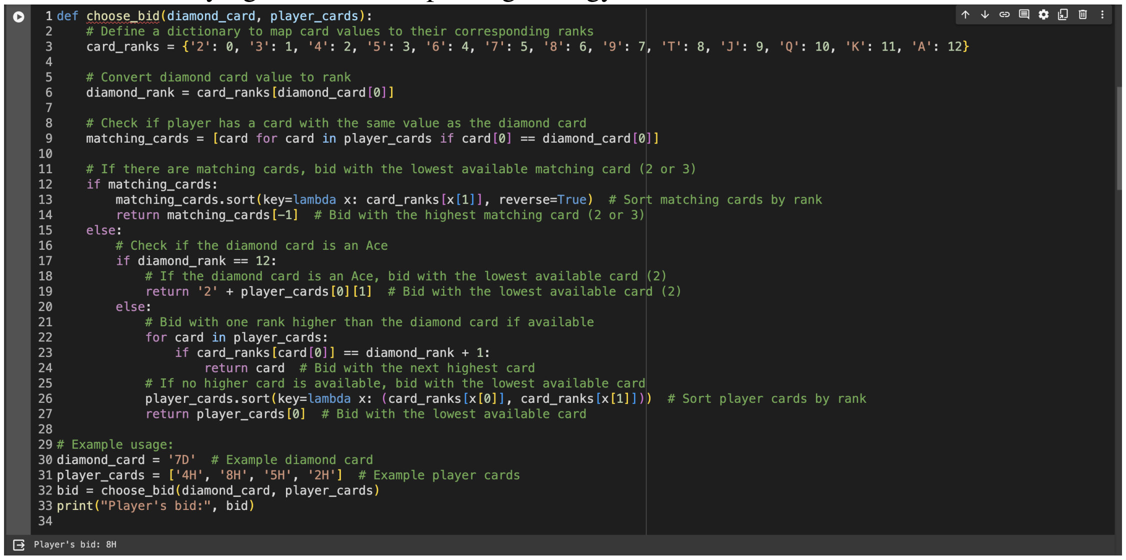
5 Code Generated

Initial Code Generated



```python
def choose_bid(diamond_card, player_cards):
    # Define a dictionary to map card values to their corresponding ranks
    card_ranks = {'2': 0, '3': 1, '4': 2, '5': 3, '6': 4, '7': 5, '8': 6, '9': 7, 'T': 8, 'J': 9, 'Q': 10, 'K': 11, 'A': 12}

    # Convert diamond card value to rank
    diamond_rank = card_ranks[diamond_card[0]]

    # Check if player has a card with the same value as the diamond card
    matching_cards = [card for card in player_cards if card[0] == diamond_card[0]]

    # If there are matching cards, bid with the lowest available matching card (2 or 3)
    if matching_cards:
        matching_cards.sort(key=lambda x: card_ranks[x[1]], reverse=True)  # Sort matching cards by rank
        return matching_cards[-1]  # Bid with the highest matching card (2 or 3)
    else:
        # If no matching cards, bid with the lowest available card
        player_cards.sort(key=lambda x: (card_ranks[x[0]], card_ranks[x[1]]))  # Sort player cards by rank
        return player_cards[0]  # Bid with the lowest available card

# Example usage:
diamond_card = '7D'  # Example diamond card
player_cards = ['4H', '8H', '5H', '2H']  # Example player cards
bid = choose_bid(diamond_card, player_cards)
print("Player's bid:", bid)
```

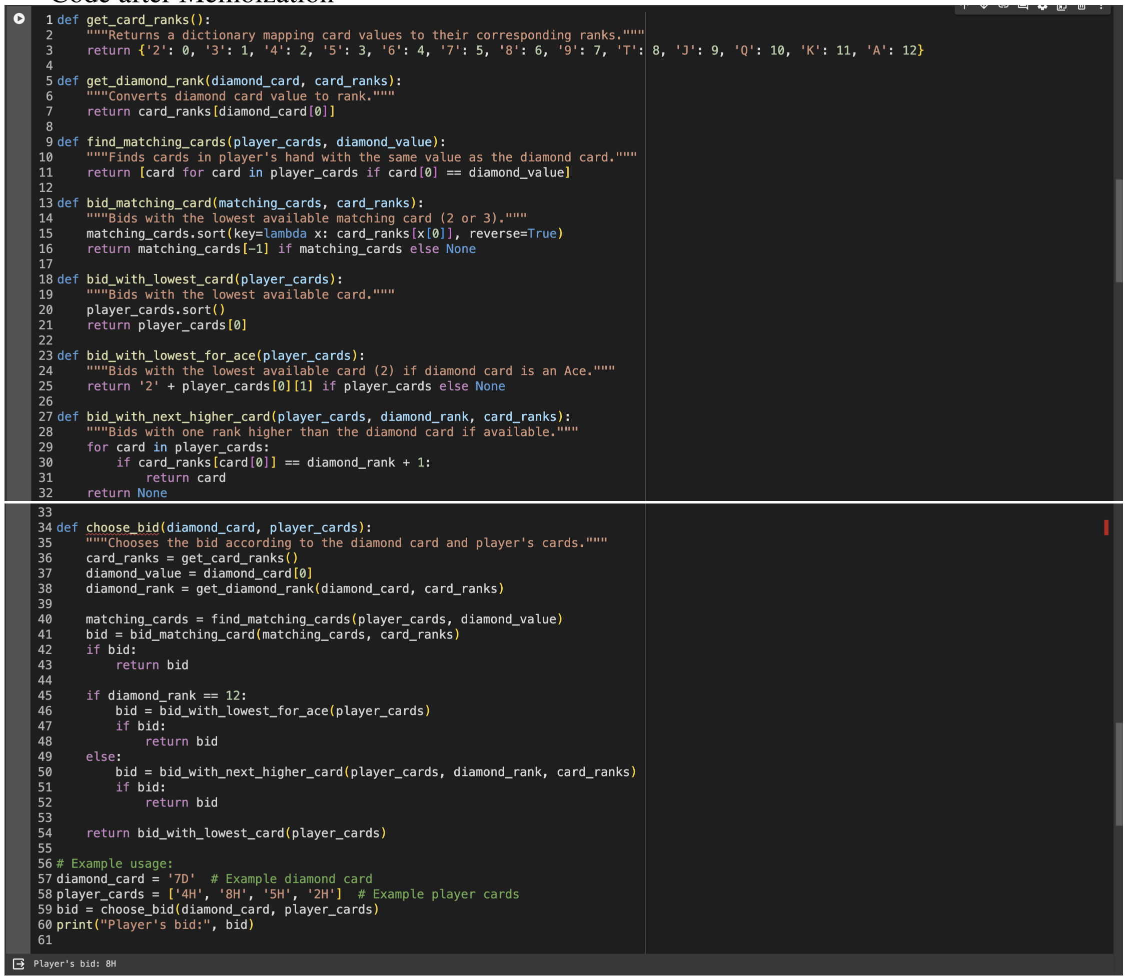
Code after Modifying Errors and Improving strategy



```python
def choose_bid(diamond_card, player_cards):
    # Define a dictionary to map card values to their corresponding ranks
    card_ranks = {'2': 0, '3': 1, '4': 2, '5': 3, '6': 4, '7': 5, '8': 6, '9': 7, 'T': 8, 'J': 9, 'Q': 10, 'K': 11, 'A': 12}

    # Convert diamond card value to rank
    diamond_rank = card_ranks[diamond_card[0]]

    # Check if player has a card with the same value as the diamond card
    matching_cards = [card for card in player_cards if card[0] == diamond_card[0]]

    # If there are matching cards, bid with the lowest available matching card (2 or 3)
    if matching_cards:
        matching_cards.sort(key=lambda x: card_ranks[x[1]], reverse=True)  # Sort matching cards by rank
        return matching_cards[-1]  # Bid with the highest matching card (2 or 3)
    else:
        # Check if the diamond card is an Ace
        if diamond_rank == 12:
            # If the diamond card is an Ace, bid with the lowest available card (2)
            return '2' + player_cards[0][1]  # Bid with the lowest available card (2)
        else:
            # Bid with one rank higher than the diamond card if available
            for card in player_cards:
                if card_ranks[card[0]] == diamond_rank + 1:
                    return card  # Bid with the next highest card
            # If no higher card is available, bid with the lowest available card
            player_cards.sort(key=lambda x: (card_ranks[x[0]], card_ranks[x[1]]))  # Sort player cards by rank
            return player_cards[0]  # Bid with the lowest available card

# Example usage:
diamond_card = '7D'  # Example diamond card
player_cards = ['4H', '8H', '5H', '2H']  # Example player cards
bid = choose_bid(diamond_card, player_cards)
print("Player's bid:", bid)
```

```
Player's bid: 8H
```

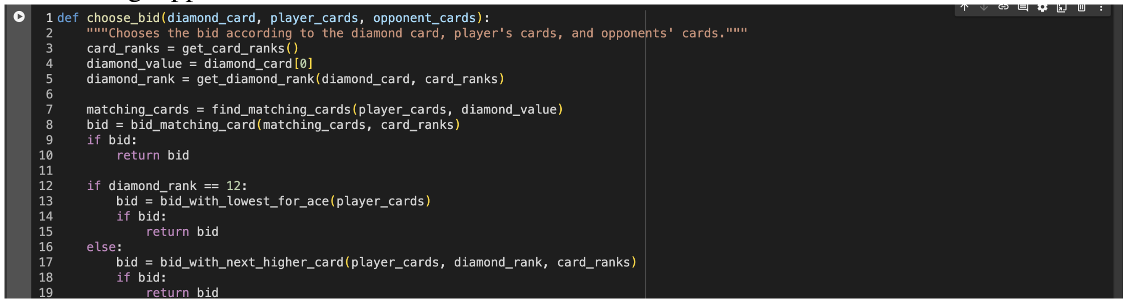
Code after Memoization



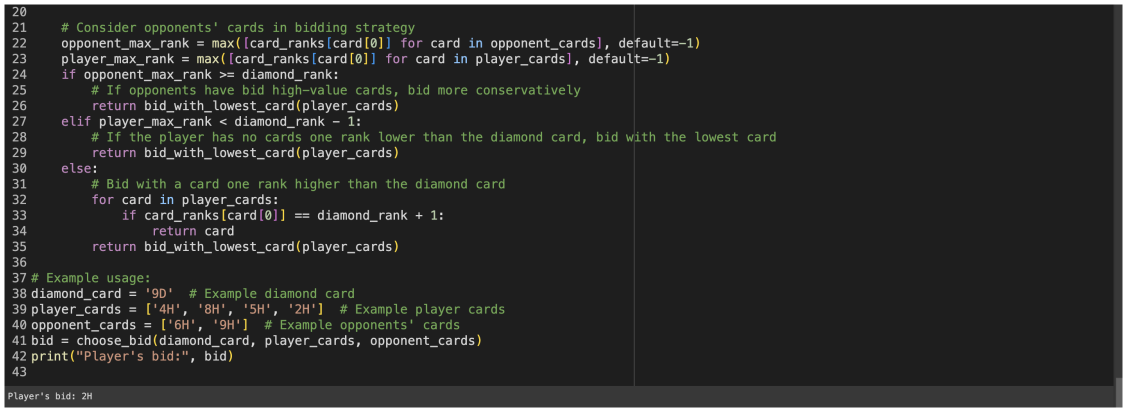
```python
def get_card_ranks():
    """Returns a dictionary mapping card values to their corresponding ranks."""
    return {'2': 0, '3': 1, '4': 2, '5': 3, '6': 4, '7': 5, '8': 6, '9': 7, 'T': 8, 'J': 9, 'Q': 10, 'K': 11, 'A': 12}

def get_diamond_rank(diamond_card, card_ranks):
    """Converts diamond card value to rank."""
    return card_ranks[diamond_card[0]]

def find_matching_cards(player_cards, diamond_value):
    """Finds cards in player's hand with the same value as the diamond card."""
    return [card for card in player_cards if card[0] == diamond_value]

def bid_matching_card(matching_cards, card_ranks):
    """Bids with the lowest available matching card (2 or 3)."""
    matching_cards.sort(key=lambda x: card_ranks[x[0]], reverse=True)
    return matching_cards[-1] if matching_cards else None

def bid_with_lowest_card(player_cards):
    """Bids with the lowest available card."""
    player_cards.sort()
    return player_cards[0]

def bid_with_lowest_for_ace(player_cards):
    """Bids with the lowest available card (2) if diamond card is an Ace."""
    return '2' + player_cards[0][1] if player_cards else None

def bid_with_next_higher_card(player_cards, diamond_rank, card_ranks):
    """Bids with one rank higher than the diamond card if available."""
    for card in player_cards:
        if card_ranks[card[0]] == diamond_rank + 1:
            return card
    return None

def choose_bid(diamond_card, player_cards):
    """Chooses the bid according to the diamond card and player's cards."""
    card_ranks = get_card_ranks()
    diamond_value = diamond_card[0]
    diamond_rank = get_diamond_rank(diamond_card, card_ranks)

    matching_cards = find_matching_cards(player_cards, diamond_value)
    bid = bid_matching_card(matching_cards, card_ranks)
    if bid:
        return bid

    if diamond_rank == 12:
        bid = bid_with_lowest_for_ace(player_cards)
        if bid:
            return bid
    else:
        bid = bid_with_next_higher_card(player_cards, diamond_rank, card_ranks)
        if bid:
            return bid

    return bid_with_lowest_card(player_cards)

# Example usage:
diamond_card = '7D'  # Example diamond card
player_cards = ['4H', '8H', '5H', '2H']  # Example player cards
bid = choose_bid(diamond_card, player_cards)
print("Player's bid:", bid)
```

```
Player's bid: 8H
```

Code Using Opponents Deck



```python
def choose_bid(diamond_card, player_cards, opponent_cards):
    """Chooses the bid according to the diamond card, player's cards, and opponents' cards."""
    card_ranks = get_card_ranks()
    diamond_value = diamond_card[0]
    diamond_rank = get_diamond_rank(diamond_card, card_ranks)

    matching_cards = find_matching_cards(player_cards, diamond_value)
    bid = bid_matching_card(matching_cards, card_ranks)
    if bid:
        return bid

    if diamond_rank == 12:
        bid = bid_with_lowest_for_ace(player_cards)
        if bid:
            return bid
    else:
        bid = bid_with_next_higher_card(player_cards, diamond_rank, card_ranks)
        if bid:
            return bid
```



```python

    # Consider opponents' cards in bidding strategy
    opponent_max_rank = max([card_ranks[card[0]] for card in opponent_cards], default=-1)
    player_max_rank = max([card_ranks[card[0]] for card in player_cards], default=-1)
    if opponent_max_rank >= diamond_rank:
        # If opponents have bid high-value cards, bid more conservatively
        return bid_with_lowest_card(player_cards)
    elif player_max_rank < diamond_rank - 1:
        # If the player has no cards one rank lower than the diamond card, bid with the lowest card
        return bid_with_lowest_card(player_cards)
    else:
        # Bid with a card one rank higher than the diamond card
        for card in player_cards:
            if card_ranks[card[0]] == diamond_rank + 1:
                return card
        return bid_with_lowest_card(player_cards)

# Example usage:
diamond_card = '9D'  # Example diamond card
player_cards = ['4H', '8H', '5H', '2H']  # Example player cards
opponent_cards = ['6H', '9H']  # Example opponents' cards
bid = choose_bid(diamond_card, player_cards, opponent_cards)
print("Player's bid:", bid)

```

```
Player's bid: 2H
```

6 Appendix

Transcripts of the chat
https://chat.openai.com/share/5ad99b20-c71d-4ca9-abcd-9db4ccbd6938