Appendix

Entire conversation/transcript:
https://chat.openai.com/share/2b581033-49fe-462c-9c75-8a4b49709b6f

ChatGPT's Initial Strategy

1. Assessing Card Values: I aimed to evaluate the point values of each diamond card and prioritize bidding for higher-ranking cards to maximize point accumulation.
2. Strategic Bidding: I strategically bid on diamond cards based on their point values and their potential impact on the overall score. I adjusted my bids according to the value of each card and the bids made by the opponent.
3. Adaptation: I adapted my bidding strategy based on the opponent's bids and the cards drawn, aiming to secure valuable diamond cards while not overspending on lower-ranking cards.
4. Risk Management: I balanced risk and reward, considering the potential point gains against the bidding costs, especially for cards with higher point values.
5. Consistency: I aimed for consistency in bidding, ensuring a steady accumulation of points throughout the game.

Reasoning for Losing the Game

1. Misjudgment of Bidding: I might have overvalued certain diamond cards or underestimated the bids of the opponent, resulting in losing bids for valuable cards.
2. Mistakes in Calculation: I might have made errors in calculating the optimal bids or assessing the point values of diamond cards, leading to suboptimal decisions.
3. Lack of Adaptation: I might have failed to adjust my bidding strategy effectively in response to the opponent's bids or changes in the game dynamics, leading to missed opportunities or inefficient use of bids.
4. Poor Risk Management: I might have taken excessive risks or failed to manage bidding resources effectively, resulting in losing bids without securing sufficient points in return.
5. Inconsistent Performance: I might have lacked consistency in my bidding strategy or made inconsistent decisions throughout the game.

Python Code: Initial Version

```
import random

def draw_card(used_cards):
    """Function to randomly draw a diamond card that hasn't been drawn before
    diamond_cards = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'Jack', 'Q
    remaining_cards = [card for card in diamond_cards if card not in used_car
    card = random.choice(remaining_cards)
    used_cards.append(card)
    return card

def get_bid(player, used_bids):
    """Function for a player to enter a bid."""
    while True:
        try:
            bid_amount = int(input(f"{player}, enter your bid: "))
            if bid_amount < 0 or bid_amount in used_bids:
                raise ValueError
            used_bids.add(bid_amount)
            return bid_amount
        except ValueError:
            print("Invalid input or bid already used. Please enter a non-nega

def determine_winner(player_bid, computer_bid, diamond_card):
    """Function to determine the winner of the bid."""
    if player_bid > computer_bid:
        print("Player wins the bid!")
        return 'player', diamond_card[1]
    elif computer_bid > player_bid:
        print("Computer wins the bid!")
        return 'computer', diamond_card[1]
    else:
        print("It's a tie! Points are divided equally.")
        return 'tie', diamond_card[1] / 2

def main():
    print("Welcome to the Diamond Game!")

    player_points = 0
    computer_points = 0
    used_cards = []
    used_player_bids = set()
    used_computer_bids = set()
    # Dictionary to map card values
    card_values = {'2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7, '8': 8, '9

    # Loop until all diamond cards have been bid on
    while len(used_cards) < 13:
        # Draw a diamond card
        diamond_card = draw_card(used_cards)
        card_value = card_values[diamond_card]  # Get the value of the drawn
        print(f"\nThe diamond card drawn is: {diamond_card}")

        # Players bid for the diamond card
        player_bid = get_bid("Player", used_player_bids)
        computer_bid = random.randint(0, 14)  # Generate a random bid for the
        while computer_bid in used_computer_bids:  # Ensure computer's bid is
            computer_bid = random.randint(0, 14)
        used_computer_bids.add(computer_bid)

        print(f"Computer bids: {computer_bid}")

        # Determine the winner of the bid
        winner, points = determine_winner(player_bid, computer_bid, (diamond_
        if winner == 'player':
            player_points += points
        elif winner == 'computer':
            computer_points += points
        else:
            player_points += points
            computer_points += points

    # Print the final scores
    print("\nGame over!")
    print(f"Player's points: {player_points}")
    print(f"Computer's points: {computer_points}")
    if player_points > computer_points:
        print("Congratulations! Player wins!")
    elif computer_points > player_points:
        print("Computer wins!")
    else:
        print("It's a tie!")

if __name__ == "__main__":
    main()
```

Python Code: After Implementing Strategy

```
import random

def draw_card(used_cards):
    """Function to randomly draw a diamond card that hasn't been drawn before
    diamond_cards = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'Jack', 'Q
    remaining_cards = [card for card in diamond_cards if card not in used_car
    card = random.choice(remaining_cards)
    used_cards.append(card)
    return card

def generate_unique_bid(used_bids):
    """Function to generate a unique bid."""
    while True:
        bid_amount = random.randint(1, 14)
        if bid_amount not in used_bids:
            used_bids.add(bid_amount)
            return bid_amount

def get_bid(player, used_bids):
    """Function for a player to enter a bid."""
    while True:
        try:
            bid_amount = int(input(f"{player},␣enter␣your␣bid:␣"))
            if bid_amount < 1 or bid_amount in used_bids:
                raise ValueError
            used_bids.add(bid_amount)
            return bid_amount
        except ValueError:
            print("Invalid␣input␣or␣bid␣already␣used.␣Please␣enter␣a␣positive

def determine_winner(player_bid, computer_bid, diamond_card):
    """Function to determine the winner of the bid."""
    if player_bid > computer_bid:
        print("Player␣wins␣the␣bid!")
        return 'player', diamond_card[1]
    elif computer_bid > player_bid:
        print("Computer␣wins␣the␣bid!")
        return 'computer', diamond_card[1]
    else:
        print("It's␣a␣tie!␣Points␣are␣divided␣equally.")
        return 'tie', diamond_card[1] / 2

def computer_strategy(diamond_card_rank, remaining_bids, player_bids, used_bi
    """Function for computer's bidding strategy."""
    # Estimate average rank of diamond cards drawn so far
    average_rank = sum(diamond_card_rank) / len(diamond_card_rank) if diamond
# Initial guess

    # Adjust bidding strategy based on average rank and player's bidding tend
    if average_rank <= 7.5:  # Adjust for low average rank
        computer_bid = random.randint(1, min(remaining_bids, 7))
# Bid conservatively
    else:  # Adjust for high average rank
        computer_bid = random.randint(1, min(remaining_bids, 14))
# Bid aggressively

    # Ensure computer's bid is higher than player's bid if possible
    if player_bids:
        max_player_bid = max(player_bids)
        computer_bid = max(computer_bid, max_player_bid + 1)

    # Ensure computer's bid is unique and not repeated
    while computer_bid in used_bids:
        computer_bid = random.randint(1, 14) # Choose a random bid if the cur

    used_bids.add(computer_bid)

    return computer_bid

def main():
    print(”Welcome␣to␣the␣Diamond␣Game!”)

    player_points = 0
    computer_points = 0
    used_cards = []
    diamond_card_rank = []
    player_bids = []
    computer_bids = set()
    remaining_bids = 13  # Total bids available

    # Loop until all diamond cards have been bid on
    while len(used_cards) < 13:
        # Draw a diamond card
        diamond_card = draw_card(used_cards)
        card_value = int(diamond_card) if diamond_card.isdigit() else 10 + ['
        print(f”\nThe␣diamond␣card␣drawn␣is:␣{diamond_card}”)

        # Player bids for the diamond card
        if len(used_cards) < 13:  # Check if there are remaining cards to bid
            player_bid = get_bid(”Player”, set(player_bids))
            player_bids.append(player_bid)

        # Computer bids for the diamond card
        computer_bid = computer_strategy(diamond_card_rank, remaining_bids, p
        print(f"Computer␣bids:␣{computer_bid}")

        # Store diamond card rank for future reference
        diamond_card_rank.append(card_value)
        remaining_bids -= 1

        # Determine the winner of the bid if the player has bid
        if len(used_cards) < 13:
            winner, points = determine_winner(player_bid, computer_bid, (diam
            if winner == 'player':
                player_points += points
            elif winner == 'computer':
                computer_points += points
            else:
                player_points += points
                computer_points += points

    # Print the final scores
    print("\nGame␣over!")
    print(f"Player's␣points:␣{player_points}")
    print(f"Computer's␣points:␣{computer_points}")
    if player_points > computer_points:
        print("Player␣wins!")
    elif computer_points > player_points:
        print("Congratulations!␣Computer␣wins!")
    else:
        print("It's␣a␣tie!")

if __name__ == "__main__":
    main()
```