Developing Strategies for the bidding card game 'Diamonds' with GenAI

Aadya Srivastava

GenAI Tool used: ChatGPT

March 26, 2024

1 Introduction

This report documents the reflections/observations made while teaching GenAI the card game "Diamonds", helping it write code for the it and come up with some winning strategies for the same. The main reason for choosing this version of the game is that GenAI seems to lacks any prior knowledge of it, leading to unbiased responses.

2 Problem Statement

The Game of Diamonds has the following rules:

- There can be 3 or 2 players.
- Each player gets a single suit of cards other than the diamond suit.
- The diamond cards are then shuffled and put on auction one by one.
- All the players must bid with one of their own cards; it is a closed bidding system, i.e., the other players don't know which card you bid until the bids are revealed.
- The banker gives the diamond card to the highest bid, i.e., the bid with the most points.
- The ranking system is: 2<3<4<5<6<7<8<9<T<J<Q<K<A
- The highest bidder gets the points of the diamond card added to their score. If there are multiple players that have the highest bid with the same card, the points from the diamond card are divided equally among them.
- The player with the most points at the end of the game wins.

3 Teaching genAI the game

Before delving into the conversation with GenAI, I attempted to understand its comprehension of "The Game of Diamonds". Ofcourse, what it understood differed from what we intended to explain. So, I initiated a new conversation thread, to prevent its prior knowledge from influencing the discussion, which is why the initial prompt clearly stated that this game was different from what it might already know.

The primary objective of the initial conversation was to explain the game's rules to GenAI and obtain a code capable of running the game for two players.

The subsequent prompts explained the rules of the game, and requested GenAI to explain its understanding, with necessary corrections provided as needed.

To assess GenAI's comprehension of the game, I asked it to provide three examples covering various scenarios that could arise during gameplay, including instances where two or three players bid cards of the same rank, resulting in point division.

While GenAI consistently covered all points outlined in the prompts and provided coherent explanations in text, its ability to implement its understanding was lacking significantly. The initial examples it provided demonstrated minimal to no understanding of the game and it only managed to provide at max two correct examples in each response.

Though, through multiple test cases and examples, along with subsequent corrections, GenAI eventually reached a point where I believed it understood the game.

4 Iterating upon strategy

When asked to come up with strategies, GenAI replied with ones that completely disregarded the rules of the game as if to just satisfy the user with some semblance of a strategy .

So, I attempted to get it to write code for games where I explain the strategy and GenAI implements it via code.

The first attempt was to just randomly selects any card to bid. This task was easily accomplished by GenAI using the random function.

After that, I tried to have it write code for a relatively simpler strategy by merely mapping the cards bid to the cards auctioned.

However, the second strategy was much more complex, and it clearly demonstrated that GenAI struggles to focus on multiple data points simultaneously. If it made a recurring mistake and was prompted multiple times, it would correct it, but it would then neglect some other point that it had gotten right the first time.

In the end, numerous corrections were needed in the code to make it run, and GenAI was also unable to determine why the computer was choosing the wrong cards to bid. When prompted, it would simply respond with phrases like "You're correct, I apologize for the confusion" or "Apologies for the oversight" and still continue to provide the same code without making any changes.

5 Reflection on Code Generated

While prompting GenAI to write code I realised that in some cases, where GenAI could make assumptions, like assuming that there should be 13 rounds because a standard diamond suit has 13 cards or for a two-player game, where it could have simply removed any suit except diamonds from the list, it instead, chose to generalise the code by saying things like, rounds should equal the number of diamond cards in the deck, rather than putting in an actual value or using list functions to remove a certain suit that wouldn't be used.

But for some details that were explicitly mentioned, like shuffling only the diamonds deck and giving the players a single suit each, it somehow assumed that the entire deck needed shuffling and distribution, possibly due to its knowledge of solitaire or other popular card games.

GenAI really tried its best to stick to its prior ideas, for example, even after giving multiple prompts in order to not make it shuffle the entire deck, it still tried to do so. Even in a new conversation thread where it was given a clear prompt and working code for a two-player game, it still attempted to change the deal_cards function twice, despite the prompt stating that the current code was correct and only the bidding strategy needed to be altered. On further prompting, it also couldn't replace the modified code with the correct version until specifically mentioned what code to replace.

6 Code

- Strategy 1: Bid the card of the same rank as the card being auctioned. (Working code in the colab link in the References section.)
- Strategy 2:
  - For cards having a value less than the Queen, bid the card which has one higher value in the hand.

    - Example 1.1: Auctioned card = 2, Computer's hand: ['2', '3', '4'], then bid 3.
    - Example 1.2: Auctioned card = 2, Computer's hand: [, '2', '5','6'], then bid 5.
  - For cards ['Q', 'K', 'A'] check if the opponent's highest card is greater or equal to yours. If yes, let them win the round by bidding the lowest card you have. Otherwise, bid the next higher value card you have in comparison to the opponents.
    - Example 2.1: Auctioned card = Q, Computer's hand: ['J', 'Q', 'K', 'A'], Player's hand: ['J', 'Q', 'K', 'A'], then bid J.
    - Example 2.2: Auctioned card = Q, Computer's hand: ['J', 'Q'], Player's hand: ['K', 'A'], then bid K.
  - Tested against a player using strategy 1.

```python
import random

# Define the ranking system
RANKS = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q', 'K', 'A']

# Define the suits
SUITS = ['Spades', 'Hearts', 'Diamonds']  # Removed 'Clubs'

# Function to create and shuffle the diamond deck
def create_diamond_deck():
    deck = [(rank, 'Diamonds') for rank in RANKS]
    random.shuffle(deck)
    return deck

# Function to deal cards to players
def deal_cards():
    player1_hand = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q', 'K', 'A']
    computer_hand = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q', 'K', 'A']
    return player1_hand, computer_hand

# Function to determine the points awarded for a drawn card
def determine_points(drawn_card):
    rank, suit = drawn_card
    return RANKS.index(rank) + 2


# Function for the bidding phase with computer strategy
def bidding_phase(player_hand, computer_hand, is_human=True, drawn_card=None):
    if is_human:
        print("Your hand:", player_hand)
        player_bid_card = input("Enter the card you want to bid (e.g., '10', 'J', 'Q', 'K', 'A'): ")
        while player_bid_card not in player_hand:
            print("Invalid input! Please choose a card from your hand.")
            player_bid_card = input("Enter the card you want to bid (e.g., '10', 'J', 'Q', 'K', 'A'): ")

        return player_bid_card
    else:
        drawn_rank = drawn_card[0]  # Rank of drawn card
        if RANKS.index(drawn_rank) < 10:  # Considering ranks 2 to Jack for lower ranking cards
```

```
            # Determine the card just higher than the drawn card
            drawn_index = RANKS.index(drawn_rank)
            # Bid the card just higher than the opponent's lowest card
            bid_card = next((rank for rank in computer_hand if RANKS.index(rank) > drawn_index), None)
            if bid_card is None:
                # If no higher rank found, bid the highest card you have in case the opponent bids low.
                bid_card = computer_hand[-1]
        else:
            # Determine the highest card in each player's hand
            opponent_max_rank = player_hand[-1]
            computer_max_rank = computer_hand[-1]

            # Bid the lowest card if opponent's highest card is higher
            if RANKS.index(opponent_max_rank) >= RANKS.index(computer_max_rank):
                bid_card = computer_hand[0]
            else:
                # Find the next higher rank from this index in computer's hand
                bid_card = next((rank for rank in computer_hand if RANKS.index(rank) > RANKS.index(opponent_max_rank)), None)

        print("Computer bids:", bid_card)
        return bid_card

# Function to play a round of the game
def play_round(player1_hand, computer_hand, draw_pile):
    # Draw the card from the draw pile
    drawn_card = draw_pile.pop()
    print("\nDrawn card from the diamond pile:", drawn_card)

    # Each player makes a bid based on the drawn card
    player1_bid = bidding_phase(player1_hand, computer_hand)

    # Computer player's bid
    computer_bid = bidding_phase(player1_hand, computer_hand, is_human=False, drawn_card=drawn_card)
    player1_hand.remove(player1_bid)
    computer_hand.remove(computer_bid)  # Remove the bid card from the computer's hand


    print("\nBids revealed!")
    print("Player 1 bids:", player1_bid)
    print("Computer bids:", computer_bid)

    # Determine points awarded for the drawn card
    points = determine_points(drawn_card)

    # Determine the winner(s) of the round
    player1_points, computer_points = calculate_points(player1_bid, computer_bid, points)

```

```
87      return player1_points, computer_points
88
89
90  # Function to calculate points and determine the winner of the
        round
91  def calculate_points(player1_bid, player2_bid, points):
92      player1_rank = player1_bid.upper()
93      player2_rank = player2_bid.upper()
94
95      # Map card names to their corresponding numerical values
96      if player1_rank in ['J', 'Q', 'K', 'A']:
97          player1_rank = RANKS.index(player1_rank) + 2
98      else:
99          player1_rank = int(player1_rank)
100
101     if player2_rank in ['J', 'Q', 'K', 'A']:
102         player2_rank = RANKS.index(player2_rank) + 2
103     else:
104         player2_rank = int(player2_rank)
105
106     if player1_rank > player2_rank:
107         print("Player 1 wins the round!")
108         player1_points = points
109         player2_points = 0
110     elif player1_rank < player2_rank:
111         print("Computer wins the round!")
112         player1_points = 0
113         player2_points = points
114     else:
115         print("It's a tie! Both players share the points.")
116         player1_points = points // 2
117         player2_points = points // 2
118
119     return player1_points, player2_points
120
121 # Main function to run the game
122 def main():
123     print("Welcome to the Diamonds card game!")
124
125     # Create and shuffle the diamond deck
126     diamond_deck = create_diamond_deck()
127
128     # Deal cards to players
129     player1_hand, player2_hand = deal_cards()
130
131     # Play rounds of the game
132     player1_score = 0
133     player2_score = 0
134     for _ in range(13):  # Fixed number of rounds
135         print("\nNew Round")
136         player1_points, player2_points = play_round(player1_hand,
                player2_hand, diamond_deck)
137         player1_score += player1_points
138         player2_score += player2_points
139         print("Player 1 score:", player1_score)
140         print("Computer score:", player2_score)
141
142     # Determine the winner of the game
```

```
    if player1_score > player2_score:
        print("Player 1 wins the game!")
    elif player1_score < player2_score:
        print("Computer wins the game!")
    else:
        print("It's a tie!")

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

Analysis and Conclusions

Explaining a card game to GenAI in theory was relatively simpler, possibly because much of the data it provided in responses was directly derived from the prompts given. However, getting it to implement its understanding, even with some simple examples, was really challenging. It was tough to get GenAI to disregard some very common assumptions about card games and to get it to focus on multiple rules and facts of the game simultaneously.

References

- ChatGPT Transcripts
  - https://chat.openai.com/share/3f271078-3075-4484-baeb-75cce99d5f45
  - https://chat.openai.com/share/2803b4a5-beb3-459f-9f05-1fd8bb7d7562
- Google Colab Link
  - https://colab.research.google.com/drive/1mtA4xoqYvdzOZlseBRHVbdaZaNnOo7y4?usp=sharing