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

Priyadarseeny Pasayat

March 26, 2024

1 Introduction

Diamonds is a trick-taking card game in which players collect diamonds — cards bearing that suit. This report explores the problem of making Generative AI(Gen AI) familiar with the unbeknownst game. The objective of the report is to measure the understanding aspect of GenAI.It aims to devise an optimized strategy to play the game.

2 Problem Statement

Despite appearances, text GenAI does not understand either the prompt written by the human or the text it generates. Every time we use a GenAI tool, we must consider its output from a skeptical perspective. It's potentially dangerous that GenAI models appear to understand the content that they use and generate, but in reality, they do not understand it. This could lead users to have misplaced trust in the GenAI output.

3 Teaching GenAI the Game

Diamonds is a two- or three-player game. In the case of two players, one suit is left out. To start the game, all diamonds are shuffled and kept face down. Each player gets all the cards from one suit. Then, the topmost diamond is drawn face up. This diamond card is the value being bidden over. Both players then decide to bid some amount and place a card with a rank equal to what they want to bid, face down. After both bids are placed, they are turned face up at the same time. The higher bid wins the diamond card being auctioned (in case of equally high bids, the diamond is equally distributed). The winner of the game is the player with a greater total face value of diamond cards won this way.

Moreover, even after understanding the rules, GenAI sometimes failed to implement them correctly while simulating a round. This could be because the model was still in the learning phase and needed more practice to get better at the game. Additionally, GenAI appeared to be confused among different variants of the Diamonds game, which could have further added to its confusion. Furthermore, GenAI seemed to forget the earlier chat history related

to the game's mechanism, such as assuming a hierarchy among the playing suits. This could be a limitation of the model's memory storage or a lack of attention to detail. Nonetheless, these difficulties highlight the challenges

4 Iterating over strategy

Here we discuss some strategies

4.1 Balancing Bidding and Trick-Taking

This approach aims to strike a balance between bidding and trick-taking. It suggests discarding a mid-range Club card (8-10) to provide a decent bid value while retaining stronger Clubs for potential trick wins.

4.2 Prioritize Winning Bids (Aggressive)t

Discard a high Club card (Queen, Jack, Ace) for a strong bid value (12-13) to secure the round if your opponent discards lower. But note that you'll lose the high card for trick-taking. This tactic is good if you have confidence in winning remaining Clubs or if winning bids is crucial for the chosen ending condition (e.g., most diamonds collected).

4.3 Prioritize Trick-Taking (Cautious)

To increase your chances of winning tricks, it's best to keep your high-value Clubs. You can discard low-value Clubs (2-4) to reduce your bid value, but still keep the strong ones for potential trick wins. This approach is particularly effective if winning tricks is crucial to the game or if you suspect your opponent might have strong Clubs.

4.4 Bidding Transparency

Both players have access to each other's entire hand, making bidding strategies more transparent. This means that you will know exactly which Clubs your opponent has and vice versa.

4.5 Focused Bidding

Knowing your opponent's Clubs allows you to adjust your bid accordingly. If they have few high Clubs, bidding with a high Club becomes a stronger move..

4.6 Reading the Opponent

To win the bid, you can try bluffing by discarding a low Club, making your hand appear weaker. But be careful, if your opponent sees through the bluff, it could backfire.

4.7

1. Start with Mid-Range: It provides a balanced approach, ensuring decent bids without sacrificing too many strong Clubs for trick-taking.

2. Adapt your strategy: Observe your opponent's behavior and adjust your strategy accordingly. You can shift to High if your opponent consistently discards low, suggesting they might prioritize trick-taking or have a weak hand. Conversely, you can shift to Low if you frequently lose bids with Mid-Range and suspect your opponent has a strong Clubs hand.

3. Keep in mind the risk: Compared to High or Bluff, Mid-Range carries less risk of losing bids due to a strong opponent hand or unsuccessful bluffing.

4. Experiment with different approaches: Remember that the best strategy ultimately depends on the specific game situation and your overall playing style. So, it's essential to try different approaches and see what works best for you.

Moving forward with Reading the opponent strategy:

```
from collections import namedtuple
import random

Card = namedtuple("Card", ["rank", "suit"])


def analyze_opponent_discard(discarded_card, revealed_diamond):
  """
  This function analyzes the opponent's discarded card.

  Args:
      discarded_card: A Card object representing the opponent's
                                              discarded card.
      revealed_diamond: A Card object representing the revealed
                                              diamond card.

  Returns:
      A string representing the inferred opponent strategy ("high
                                              ", "mid_range", "low",
                                              "uncertain").
  """
  opponent_strategy = "uncertain"

  # Analyze discard rank based on revealed diamond rank (higher
                                          diamond suggests less value
                                           on own suit)
  diamond_value = {"A": 14, "K": 13, "Q": 12, "J": 11, "10": 10, "
                                          9": 9, "8": 8, "7": 7, "6":
                                           6, "5": 5, "4": 4, "3": 3,
                                           "2": 2}
  discarded_value = diamond_value.get(discarded_card.rank, 0)
  revealed_value = diamond_value.get(revealed_diamond.rank, 0)

  if discarded_value > revealed_value:
    # Discard higher than revealed diamond suggests prioritizing
```

```
                                        bids
        opponent_strategy = "high"
    elif discarded_value == 0 or discarded_value < revealed_value //
                                        2:
        # Discard lower than half the revealed diamond's value
                                        suggests prioritizing
                                        trick-taking or
                                        uncertainty
        opponent_strategy = "low"
    else:
        # Mid-range discard is inconclusive

    return opponent_strategy


def choose_discard(my_clubs, revealed_diamond, strategy,
                                    opponent_discard):
    """
    This function chooses a card to discard for bidding based on the
                                        strategy and opponent
                                        analysis.

    Args:
        my_clubs: A list of Card objects representing your Clubs.
        revealed_diamond: A Card object representing the revealed
                                        diamond card.
        strategy: A string representing your chosen strategy ("high
                                        ", "mid_range", "low",
                                        "bluff").
        opponent_discard: A string representing the inferred
                                        opponent strategy based
                                        on their discard
                                        analysis.

    Returns:
        A Card object representing the chosen card to discard.
    """
    adjusted_strategy = strategy

    # Potentially adjust strategy based on opponent analysis (be
                                        more aggressive if opponent
                                        seems weak)
    if opponent_discard == "low" and strategy in ["mid_range", "
                                        bluff"]:
        adjusted_strategy = "high"  # Become more aggressive if
                                        opponent prioritizes
                                        trick-taking

    # Call the original choose_discard logic with potentially
                                        adjusted strategy
    return original_choose_discard(my_clubs.copy(), revealed_diamond
                                        , adjusted_strategy)
```

```
def original_choose_discard(my_clubs, revealed_diamond, strategy):
    """
    This function chooses a card to discard for bidding based on the
                                            strategy.

    Args:
        my_clubs: A list of Card objects representing your Clubs (e.
                                                g., Card(rank="J", suit
                                                ="Clubs")).
        revealed_diamond: A Card object representing the revealed
                                                diamond card.
        strategy: A string representing the chosen strategy ("high",
                                                 "mid_range", "low", "
                                                bluff").

    Returns:
        A Card object representing the chosen card to discard.
    """

    if strategy == "high":
        # Prioritize winning bids - discard the highest Club
        return max(my_clubs, key=lambda card: card.rank)
    elif strategy == "mid_range":
        # Balance bidding and trick-taking - discard a mid-range Club
        return my_clubs[len(my_clubs) // 2]  # Pick the middle card
    elif strategy == "low":
        # Prioritize trick-taking - discard the lowest Club
        return min(my_clubs, key=lambda card: card.rank)
    elif strategy == "bluff":
        # Bluffing tactic - discard a random non-high Club (if
                                                available)
        high_clubs = ["A", "K", "Q", "J"]
        candidates = [card for card in my_clubs if card.rank not in
                                                high_clubs]
        if candidates:
            return random.choice(candidates)
        else:
            # Fallback to mid-range if no non-high Clubs available
            return my_clubs[len(my_clubs) // 2]
```

5 Path Forward

In conclusion, teaching GenAI to play the bidding game of Diamonds was a challenging task. While the model showed potential in understanding the game's rules, it still struggled with implementing the rules correctly during simulations. Moreover, the model appeared to be confused among different variants of the game and could not retain important details discussed earlier.

To improve GenAI's performance, it is recommended to provide the model with more training data and a diverse set of game variants to learn from. Additionally, implementing a memory mechanism in the model could help it retain important details from previous rounds

and improve its decision-making ability.

Overall, GenAI has the potential to become a proficient player in the bidding game of Diamonds, but it requires additional training and improvements to overcome its current limitations.

A Appendix