Women Engineer's program Gen AI Assignment-2

Developing strategies for the bidding card game 'Diamonds' with Gen AI

Gen AI tool used: Gemini

March 26, 2024

1 Introduction

The game of 'Diamonds' presents a compelling challenge involving three players. Initially introduced to the WE scholars during a bootcamp led by our program dean, Asokan Pichai, this report documents the process of harnessing Gen AI to develop winning strategies, integrate it as a third player, and establish a testing methodology for the code.

2 Problem statement

The game of Diamonds has the following rules:-

- Each player gets a 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 face down.
- The banker gives the diamond card to the highest bid, i.e. the bid with the most points.
- Point hierarchy: 2<3<4<5<6<7<8<9<T<J<Q<K<A
- The winning player gets the points of the diamond card to their column in the table.
- 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 wins at the end of the game.

3 Teaching Gen AI the game

Initiating dialogue with Gemini, I proposed the idea of engaging in a game based on provided rules. Gemini responded by seeking clarification on the total number of players involved. Upon confirming three players, I received the following response:

"Great! Let's play a round of this diamond bidding game with 3 players. I'll be one of the players. We'll need a way to simulate dealing the cards and bidding anonymously.

Here's an option:

- Dealing the cards: You can tell me which suit (Spades, Hearts, or Clubs) each player gets. I can't see your cards, so you'll need to keep track of them yourself.
- Bidding: For each diamond card being auctioned, tell me the value of the card you want to bid (e.g., "7 of Clubs"). I will do the same, keeping my bid hidden from you.
- Resolving the bid: You can then tell me the highest bid value and how many players tied for the highest bid. Based on that, I can tell you who wins the diamond card and any points awarded.

Does this sound like a workable way to play?"

However since the task was primarily to get GenAI to build winning strategies for the game I proceeded to do that.

4 Iterating upon strategy

When I asked GenAI to produce a few strategies for playing the game it displayed an impressive understanding of the game. This was the response:

"Sure, here are a few strategies you can consider for the diamond bidding game:

- Balance Point Preservation vs. Diamond Acquisition: In the early stages, you might want to prioritize holding onto high-value cards in your suit. These can be crucial for winning later auctions with valuable diamonds. However, don't be afraid to sacrifice a lower value card to secure a strong diamond, especially if you suspect others might do the same. Reading Your Opponents:
- Observe Bidding Patterns: Pay attention to how your opponents bid on different diamond values. Are they consistently aggressive for high-value diamonds, or do they hold back? This can help you predict their future bids and adjust your strategy accordingly.
- Watch Out for Bluffs: If someone consistently bids high on low-value diamonds, they might be bluffing to scare others away from bidding on more valuable ones later. Bidding Strategies:
- Incremental Bidding: Don't reveal your hand too early. Start with lower value bids and gradually increase them if needed. This helps you avoid overpaying and keeps your opponents guessing.
- Sacrifice Plays: Sometimes, it might be strategic to sacrifice a high-value card from your suit to win a particularly valuable diamond, especially if you think others might bid heavily for it.
- Adapt Based on the Situation: There's no single "winning" strategy. Be flexible and adapt your approach based on the flow of the game, the revealed diamonds, and your remaining cards.
- Consider the Overall Point Spread: Don't get too focused on winning every single diamond. Keep an eye on your opponents' point totals and prioritize bids that could significantly increase your lead or prevent them from catching up.

These are just a few ideas to get you started. As you play more games, you'll develop your own strategies based on your play style and experience!"

I attempted to put the strategies into code by giving the following prompt: "How will you approach the task to write a code using these strategies that acts like the 3rd player? "

To this prompt I received a nicely put together strategy explaining the different components and strategies that it would include. I saved this particular response in a text file and proceeded to prompt Gemini with parts of this response to get the code for each component and strategy.

5 Code

```
from collections import Counter

class Hand:
  """
  This class represents a player's hand in the diamond bidding game.
  """
  def __init__(self, suit):
    """
    Initializes the hand with a specific suit using a while loop.
    """
    self.suit = suit
    self.cards = Counter()
    rank = 2
    while rank <= 14:
      self.cards[str(rank)] = 1  # Add each card value (2-A)
      rank += 1
    self.revealed_diamonds = []  # Track revealed diamond values
    self.opponent_history = {}  # Optional: Track opponent bids (dictionary)

  def remove_card(self, card_value):
    """
    Removes a card from the hand if present.
    """
    if str(card_value) in self.cards:
      self.cards[str(card_value)] -= 1
      if self.cards[str(card_value)] == 0:
        del self.cards[str(card_value)]  # Remove card if count reaches 0

  def get_remaining_cards(self):
    """
    Returns a list of remaining card values in the hand.
    """
    return list(self.cards.keys())

  def has_card(self, card_value):
    """
    Checks if the player has a specific card value in their hand.
    """
    return str(card_value) in self.cards and self.cards[str(card_value)] > 0

  def update_revealed_diamond(self, diamond_value):
    """
    Updates the list of revealed diamond values.
    """
    self.revealed_diamonds.append(diamond_value)

  def update_opponent_bid(self, opponent_name, diamond_value, bid_value):
    """
    Updates the opponent’s bidding history (optional).
    """
    if opponent_name not in self.opponent_history:
      self.opponent_history[opponent_name] = []
    self.opponent_history[opponent_name].append((diamond_value, bid_value))

  def place_bid(self, diamond_value):
    """
    Places a bid for a diamond based on a bidding strategy.
    """
    # Minimum bid based on diamond value
    min_bid = diamond_value // 2

    # Prioritize high cards in the beginning (adjustable threshold)
    if len(self.get_remaining_cards()) > 5 and diamond_value < 8:
      return min_bid

    # Increase bid slightly for higher value diamonds
    if diamond_value >= 10:
      min_bid += 1

    # Adjust based on remaining high cards (optional)
    high_cards_remaining = sum(self.cards[card_value] for card_value in
    [’J’, ’Q’, ’K’, ’A’])
    if high_cards_remaining > 2 and diamond_value >= 8:
      min_bid += 1
      # Be more willing to bid for high value diamonds if high cards remain

    # (Optional) Adjust based on opponent history (basic example)
    if self.opponent_history:
      # Consider recent opponent bids for similar diamond values
      # (This is a simplified approach, more complex analysis is possible)
      for opponent, bids in self.opponent_history.items():
        recent_bids = bids[-2:]  # Consider the last 2 bids
        if any(value >= diamond_value // 2 for value, _ in recent_bids):
          min_bid += 1
          # Increase bid if opponent recently bid high for similar values

    return min_bid
# Example usage
hand = Hand("Spades")
print(hand.get_remaining_cards())
# Output: [’2’, ’3’, ’4’, ’5’, ’6’, ’7’, ’8’, ’9’, ’10’, ’J’, ’Q’, ’K’, ’A’]

hand.remove_card(7)
print(hand.get_remaining_cards())
# Output: [’2’, ’3’, ’4’, ’5’, ’6’, ’8’, ’9’, ’10’, ’J’, ’Q’, ’K’, ’A’]

print(hand.has_card(7))  # Output: False

# Example usage
hand = Hand("Spades")
hand.update_revealed_diamond(8)
hand.update_revealed_diamond(10)
print(hand.revealed_diamonds)  # Output: [8, 10]

# Example usage
hand = Hand("Spades")
hand.update_revealed_diamond(8)
print(hand.place_bid(5))  # Output: 2 (Minimum bid)
print(hand.place_bid(10))  # Output: 6 (Increased for higher value)

# Example usage
hand = Hand("Spades")
hand.update_revealed_diamond(8)
print(hand.place_bid(5))  # Output: 2 (Minimum bid)
print(hand.place_bid(10))  # Output: 6 (Increased for higher value)
print(hand.place_bid(8))  # Output: 4 (Prioritize high cards if many remain)
hand.remove_card(’K’)
print(hand.place_bid(8))
# Output: 5
(More willing to bid for high value diamonds after losing a high card)
# Example usage
hand = Hand("Spades")
hand.update_revealed_diamond(8)
print(hand.place_bid(5))  # Output: 2 (Minimum bid)

# Simulate opponent bids (assuming opponent names are known)
hand.update_opponent_bid("Player1", 8, 4)
hand.update_opponent_bid("Player2", 5, 1)
print(hand.place_bid(8))  # Output: 4 (Prioritize high cards if many remain)
```

6 Analysis and Conclusion

As mentioned in one of the previous sessions, explaining a card game is one of the hardest technical problems. This is why I was really impressed by the response I received from Gemini. However, some part of the code generated wasn't up to the mark and needed some modifications. Overall it was a good experience and a pretty successful one at that.