Teaching genAI to Play Diamonds: A Journey in Strategic Optimization

Introduction

In this report, I'll detail the journey of teaching genAI, how to play the intriguing card game of Diamonds. Our objective was to equip genAI with an optimizing strategy, allowing it to compete effectively against human players. This endeavor involved breaking down the game's rules, discussing strategic approaches, and guiding genAI in implementing these strategies.

Rules of the Game

Firstly, let's get acquainted with the rules of Diamonds. It's a card game where players bid for diamond cards, each carrying a specific point value. The goal is simple: amass the highest score by winning bids and collecting points from diamond cards. In Diamonds, players use a standard 52-card deck. Each round, players bid with cards from their hand to win diamond cards, accumulating points. The twist? Players can bid with cards of any suit, adding strategic depth. The player with the highest score at the end wins.

Methodology

Our Approach

Teaching genAI wasn't just about feeding it information. It was a process of back-and-forth, understanding, and refinement. We began by introducing genAI to the game's fundamentals, emphasizing bidding tactics and point calculations. Then, we delved into strategic discussions, exploring different bidding strategies and their implications.

As we progressed, we realized the importance of translating these strategies into code. We brainstormed on how to represent bidding tactics and decision-making processes algorithmically. Through iterative coding and testing, we refined genAI's gameplay algorithms, tweaking parameters and logic to enhance its strategic capabilities. This iterative approach allowed us to continuously improve genAI's performance and adaptability in playing Diamonds.

Reflections on conversation with genAI & prompting/Learnings

Strategic Discussions

Our conversations with genAI revolved around strategic considerations. We discussed the importance of maximizing point accumulation while managing card resources effectively. For instance, we debated when to bid aggressively to secure high-value diamond cards and when to bid conservatively to preserve crucial cards for future bids.

Reflections on code that was generated/Snippets

The crux of our endeavor lay in translating these discussions into code. We prompted genAI to generate code implementing the discussed strategies. It was a process of trial and error, refining the code based on feedback and iteratively improving its gameplay algorithms.

Throughout the coding process, we encountered various challenges, such as optimizing decision-making algorithms and handling edge cases. We experimented with different approaches, leveraging genAI's ability to analyze game states and make informed decisions. Collaborative problem-solving and continuous refinement were key to overcoming these challenges and enhancing genAI's gameplay capabilities.

Strategy in Action

With the strategy coded into genAI, it was time for practical testing. Playing against genAI revealed its competitive edge. It adeptly simulated the auction process, evaluated bids, and made optimal decisions based on available information. Its adaptive decision-making allowed it to adjust bid strategies based on the evolving game state.

Conclusion

In conclusion, teaching genAI to play Diamonds was an enlightening experience. It showcased the potential of AI in mastering complex games through strategic optimization. Moving forward, further refinement of genAI's gameplay algorithms holds promise for its competitiveness against human players and opens doors to broader applications in AI game strategy.

Appendices

GenAI transcripts at :

ChatGPT Transcripts
https://chat.openai.com/share/c0382b25-8842-4518-b421-66c1d21e20f8

Code:

Listing 1: Optimized Code

```
import random

# Define card values
card_values = {'2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7, '8': 8, '9': 9, "

# Define the number of players
num_players = 4  # Change this number as per your preference

# Initialize players and their hands
players = []
for i in range(num_players):
    players.append({'hand': list(card_values.keys()), 'points': 0})

# Initialize a list to hold the diamond cards
diamond_cards = list(card_values.keys())

# Shuffle the diamond cards
random.shuffle(diamond_cards)

# Start the auction loop
for diamond_card in diamond_cards:
    print("\nCurrent diamond card:", diamond_card)

    # Initialize a dictionary to hold bids from players
    bids = {}

    # Deal one card to each player
    for player in players:
        player['hand'].append(diamond_card)

    # Accept bids from players
    for i, player in enumerate(players):
        print(f"\nPlayer {i+1}'s hand:", player['hand'])
        bid_card = input(f"Player {i+1}, place your bid (select one card from yo

        # Check if the bid card is valid
        if bid_card not in player['hand']:
            print("Invalid bid! You don't have that card.")
            continue

        # Remove the bid card from the player's hand
        player['hand'].remove(bid_card)
```

```
            # Record the bid
            bids[i] = bid_card

        # Determine the winning bid
        highest_bid_value = max([card_values[bid] for bid in bids.values()])
        winners = [player for player, bid in bids.items() if card_values[bid] == hig

        # Award the diamond card points to the winning bidder
        points_to_award = card_values[diamond_card]
        winning_player = winners[0]  # Select the first winner
        players[winning_player]['points'] += points_to_award

        print(f"\nWinning bid(s): Player {winning_player + 1}")
        print(f"Points awarded: {points_to_award}")

# Calculate scores and determine the winner
player_scores = [(i+1, player['points']) for i, player in enumerate(players)]
player_scores.sort(key=lambda x: x[1], reverse=True)
print("\nFinal Scores:")
for player, score in player_scores:
    print(f"Player {player}: {score} points")

winner = player_scores[0][0]
print(f"\nPlayer {winner} wins!")
```