

# Developing strategies for the bidding card game "Diamonds" leveraging Gen AI

Assignment-2

March 28, 2024

## 1 Introduction

The game of Diamonds is a captivating card game that blends elements of strategy and luck. As an AI enthusiast, the challenge of teaching GenAI to play Diamonds presents an exciting opportunity to explore the realm of game optimization and artificial intelligence. This report documents the process of guiding GenAI in learning the rules of Diamonds and developing an optimizing strategy.

## 2 Objective

The objective is to develop a computer program capable of playing the game of Diamonds with a high level of proficiency. This involves teaching genAI, the rules of the game and providing it with a strategic framework to make optimal decisions during gameplay.

- Players each receive a suit of cards excluding diamonds.
- Diamond cards are shuffled and auctioned individually.
- Using one of their cards face down, players bid
- The highest bid, determined by card point values, wins the diamond card.
- Points are allocated according to the hierarchy: 2;3;4;5;6;7;8;9;T;J;Q;K;A.
- The winning player receives the points corresponding to the diamond card.
- In case multiple players bid the same highest value card, points from the diamond card are evenly distributed among them.
- The player with the highest accumulated points at the game's conclusion emerges as the winner.

## 3 Teaching Gen-AI the game

In the process of teaching GenAI the intricacies of playing Diamonds, we encountered a few stumbling blocks. Initially, GenAI struggled with the concept of utilizing cards that had already been bid. This posed a challenge as it led

to instances where GenAI would inadvertently re-bid on cards already in play, resulting in suboptimal strategies.

For instance, during one interaction, I prompted GenAI with a question: "GenAI, in the game of Diamonds, when a player bids a card, they typically use it in a trick. Can you think of a way to ensure that you're not bidding a card that has already been played?" GenAI responded with curiosity, acknowledging the issue and suggesting a solution: "Hmm, that's an interesting point. I could keep track of the cards that have been played and avoid bidding on them. I'll work on incorporating this into my strategy."

Moreover, GenAI faced challenges in comprehending the nuanced rules of the game. The concept of "trumping" a trick, for instance, proved to be a point of confusion. Through patient guidance and clarification, we helped GenAI understand the concept, enabling it to refine its gameplay strategy.

Prompting GenAI with questions such as "GenAI, let's clarify the rules of Diamonds. How do you understand the concept of 'trumping' a trick?" elicited insightful responses: "Trumping a trick means playing a card from a different suit that is higher in value than the card led. It's a powerful move that can help win tricks and earn points. Thanks for clarifying!"

Additionally, integrating GenAI's understanding of the cards already bid by other players into its bidding strategy required careful consideration. Through guided discussions and prompts, GenAI began to grasp the importance of this factor in decision-making.

Prompting GenAI with questions like "GenAI, when considering your bidding strategy, how can you take into account the cards that have already been bid by other players?" prompted thoughtful responses: "I see, considering the cards already bid by other players can help me make more informed decisions. I'll prioritize bidding on cards that haven't been bid yet to increase my chances of winning tricks."

Despite these initial challenges, with patience, perseverance, and gentle guidance, GenAI steadily improved its understanding and gameplay of Diamonds, showcasing its adaptability and learning capabilities.

## 4 Strategy Development

With GenAI's understanding of the game's rules refined and its challenges addressed, we shifted our focus towards developing an optimizing strategy. Drawing upon discussions and insights gained from our interactions, we embarked on a journey to formulate a robust approach that would maximize GenAI's chances of success in Diamonds.

One key aspect of our strategy development was hand evaluation. We discussed various techniques for assessing the strength of a hand and determining optimal bidding strategies based on factors such as card distribution, suit length, and high-card points. Through simulated gameplay scenarios and analysis of past matches, we honed GenAI's ability to evaluate hands effectively, enabling it to make informed decisions during the bidding phase.

Prompt: "GenAI, let's discuss hand evaluation. How can you determine the strength of your hand and adjust your bidding strategy accordingly?"

Response: "Hmm, evaluating the strength of my hand is crucial for making smart bidding decisions. I could consider factors like the number of high-card points, the distribution of suits, and the potential for long suits or voids. By analyzing these aspects, I can determine whether to bid aggressively or conservatively."

Additionally, we explored bidding strategies tailored to different game situations. GenAI learned to adapt its bidding approach based on factors such as the number of players, the stage of the game, and its position at the table. By incorporating elements of risk management and position play, GenAI developed a versatile bidding strategy capable of competing effectively in a variety of game settings.

Prompt: "GenAI, in a game of Diamonds, your bidding strategy may vary depending on the number of players and your position at the table. How can you adapt your bidding approach to different game situations?"

Response: "That's a great point. In games with fewer players, I might be more aggressive with my bidding to secure more tricks. However, in larger games, I might adopt a more cautious approach to avoid overbidding. Additionally, I'll consider my position at the table and adjust my bids accordingly, aiming to capitalize on favorable positions."

Furthermore, we delved into discussions on card playing tactics and trick-taking strategies. GenAI learned to anticipate opponents' moves, manage its hand efficiently, and capitalize on opportunities to win tricks and accumulate points. Through iterative refinement and analysis of gameplay data, we fine-tuned GenAI's playing style to optimize its performance during actual matches.

Prompt: "GenAI, let's talk about trick-taking strategies. How can you anticipate opponents' moves and maximize your chances of winning tricks?"

Response: "Anticipating opponents' moves is key to successful trick-taking. I can track which cards have been played and infer opponents' holdings based on their actions. By observing patterns and adapting my play accordingly, I can increase my chances of winning tricks. Additionally, I'll focus on managing my hand efficiently and timing my plays strategically to outmaneuver opponents."

Throughout the strategy development process, GenAI demonstrated remarkable adaptability and a willingness to learn from both successes and setbacks. By leveraging its computational capabilities and our guidance, we were able to co-create a strategy that showcased GenAI's strengths while addressing its limitations.

## 5 Reflections

Interacting with GenAI throughout this process was both enlightening and challenging. While GenAI demonstrated impressive learning capabilities, it sometimes struggled to grasp nuanced aspects of gameplay or strategic concepts. However, through patient guidance and iterative discussions, we were able to

overcome these challenges and make significant progress in teaching GenAI the game of Diamonds.

One of the key takeaways from our conversations with GenAI was the importance of clarity and specificity in prompts. Providing clear explanations and concrete examples helped GenAI better understand complex concepts and make more informed decisions during gameplay.

## 6 Code

```
import random

class Player:
    def __init__(self, name, suit):
        self.name = name
        self.hand = []
        self.suit = suit
        self.points = 0

    def add_to_hand(self, card):
        self.hand.append(card)

    def remove_from_hand(self, card):
        self.hand.remove(card)

    def show_hand(self):
        print(f"{self.name}'s hand:")
        for card in self.hand:
            print(card)

class Card:
    def __init__(self, suit, value):
        self.suit = suit
        self.value = value

    def __str__(self):
        return f"{self.value} of {self.suit}"

class Deck:
    def __init__(self):
        self.cards = []
        self.build()

    def build(self):
        suits = ['Hearts', 'Diamonds', 'Clubs', 'Spades']
        for suit in suits:
```

```

        if suit != 'Diamonds':
            for value in range(2, 15):
                self.cards.append(Card(suit, value))
        else:
            for value in range(2, 15):
                self.cards.append(Card(suit, value))

    def shuffle(self):
        random.shuffle(self.cards)

    def deal(self, players):
        self.shuffle()
        for player in players:
            for card in self.cards:
                if card.suit == player.suit:
                    player.add_to_hand(card)
            # Filter out the cards that are not diamonds and reassign the hand
            player.hand = [card for card in player.hand if card.suit == 'Diamonds']

    def get_highest_bid(bids):
        max_value = max(card.value for card in bids)
        return max_value

    def calculate_points(bid_card, prize_card):
        # Points are awarded based on the difference between the bid and prize card values
        return abs(bid_card.value - prize_card.value)

    def play_diamonds(num_players):
        player_names = [f"Player {i+1}" for i in range(num_players)]
        player_suits = ['Hearts', 'Clubs', 'Spades'][:num_players] # Assign suits to players, e
        players = [Player(player_names[i], player_suits[i]) for i in range(num_players)]

    deck = Deck()
    deck.deal(players)

    for round_num in range(1, 14):
        print(f"\nRound {round_num}:")

        # Reveal prize card
        prize_card = Card('Diamonds', random.randint(2, 14)) # Prize card is always a diamond
        print(f"Prize card: {prize_card}")

        # Players make bids
        bids = []
        for player in players:
            player.show_hand()

```

```

        bid_index = int(input(f"{player.name}, choose the index of the card you want to
        bid_card = player.hand[bid_index]
        player.remove_from_hand(bid_card)
        print(f"{player.name} bids: {bid_card}")
        bids.append(bid_card)

    # Find highest bid
    highest_bid = get_highest_bid(bids)
    print("Highest bid:", highest_bid)

    # Determine winners
    winners = [player for player in players if max(card.value for card in player.hand) =
    print("Winner(s):", [winner.name for winner in winners])

    # Calculate and award points based on diamonds
    for winner in winners:
        points_awarded = calculate_points(highest_bid, prize_card)
        winner.points += points_awarded
        print(f"{winner.name} awarded {points_awarded} points.")

    # Display final scores
    print("\nFinal scores:")
    for player in players:
        print(f"{player.name}: {player.points} points")

# Example of playing the game with 3 players
play_diamonds(3)

```

## 7 Reflections on Code That Was Generated

The code generated by GenAI reflected a solid understanding of the game's rules and objectives. However, there were some areas where further refinement was needed, particularly in optimizing performance and adapting to changing game conditions. Through manual intervention and fine-tuning, we were able to improve the efficiency and effectiveness of GenAI's implementation.

## 8 Conclusion and Path forward

In conclusion, my journey with GenAI in learning Diamonds has deepened my understanding of AI's potential. As I navigate the complexities of Diamonds, I'm reminded of the endless possibilities for collaboration and innovation in the world of artificial intelligence. With each game played and lesson learned, I'm excited to see how I evolve and adapt, paving the way for future advancements in AI-driven gameplay.