Developing Strategies for the Bidding Card Game “Diamonds” with ChatGPT

Misha Jain

March 25, 2024

Abstract

This report explores the development of strategies for the bidding card game “Diamonds” using ChatGPT. We discuss the objectives, problem statement, teaching ChatGPT the game, iterative strategy development, analysis, and conclusions.

1 Introduction

The card game “Diamonds” is all about clever bidding and playing the right cards at the right time. In this report, our goal is to help an artificial intelligence called ChatGPT learn how to play “Diamonds” really well. We’ll start by teaching ChatGPT the basic rules of the game. Our aim is to turn ChatGPT into a top-notch player that can make smart decisions when bidding for cards.

2 Objective

The objective of this project is to develop an AI agent capable of playing the “Diamonds” card game with human-like intelligence. This involves teaching ChatGPT the rules of the game, implementing a bidding strategy, and refining the strategy through iterative development.

3 Problem Statement

The “Diamonds” card game follows specific rules, as described in the provided document. Players bid with cards from their hand to acquire diamond cards, earning points based on the value of the diamonds. Rules of the game are -

1. Getting Ready: Each player gets a bunch of cards, but not the diamond ones.
2. Time to Bid: One by one, diamond cards are shown, and players bid with their cards to win them.
3. Winning: The player with the best card wins the diamond card. If some players have equally good cards, they share the diamond.
4. Getting Points: The winner adds up points from their diamond cards.
5. Game Over: Keep going until all diamond cards are gone. The player with the most points wins.

The challenge lies in teaching ChatGPT the rules of the game and developing a bidding strategy that maximizes its chances of winning.

4 Teaching ChatGPT the Game

To teach ChatGPT the rules of the game, I break down each rule into simple steps and provide examples where necessary. Here's how I explain each rule to ChatGPT:

1. **Getting Ready:** I explain to ChatGPT that at the beginning of the game, each player receives a set of cards, but none of them are diamond cards. For example, if there are four players, each player gets all the cards from one of the non-diamond suits, like hearts, clubs, or spades.
2. **Time to Bid:** ChatGPT learns that during the game, diamond cards are revealed one by one, and players bid with their cards to win them. For instance, if ChatGPT randomly reveals a diamond 10, and a player chooses a 5 of hearts, the player wins if their card is greater.
3. **Winning:** ChatGPT understands that the player with the highest-ranking card wins the diamond card. If multiple players have equally high-ranking cards, they share the diamond. For example, if ChatGPT reveals a 2 of spades and a player bids with a 5 of hearts, and the player's card is greater, they receive the points from the diamond card.
4. **Getting Points:** ChatGPT learns that the winner of each diamond card adds up points based on the card's value. For example, if the player's card was greater and they win a diamond 10, they receive 10 points.
5. **Game Over:** ChatGPT knows that the game continues until all diamond cards are won. The player with the most points at the end wins the game. For instance, if ChatGPT reveals a diamond card one by one, the points are accumulated after each round. When prompted, ChatGPT provides the final scores of all players.

5 Iteration upon the Code Strategy

Upon playing an example game with ChatGPT, I prompted it to provide an approach related to the coding part in Python. It provided the following initial code:

```
class Card:
    def __init__(self, rank, suit):
        self.rank = rank
        self.suit = suit
        # Add methods if needed

class Player:
    def __init__(self, name, cards):
        self.name = name
        self.cards = cards
        self.score = 0
        # Add methods if needed

class Game:
    def __init__(self, players):
        self.players = players
        # Initialize deck, assign cards to players, etc.

    def start_auction(self):
        # Auction logic here

    def update_scores(self, winner, points):
        # Update scores based on the winner of the auction and the points earned
    def end_game(self):
        # Determine the winner and end the game

# Main function to run the game
def main():
    # Initialize players, deck, etc.
    # Create a Game instance
    # Start the game loop

if __name__ == "__main__":
    main()
```

Upon asking for the pseudocode, ChatGPT provided the following:

```
Initialize game:
    Create deck of cards (excluding diamonds)
    Assign each player a suit of cards
    Set initial scores for each player to 0

Start auction phase:
    Shuffle diamond cards
    For each diamond card in the shuffled deck:
        Display the current diamond card
        For each player:
            Get bid from player
            Validate bid
        Determine highest bidder(s)
        Award points to winning bidder(s) based on diamond card value
        Update scores
        Display round winner(s) and points earned

End game:
    Display final scores
    Determine winner based on total points
    Display winner
```

After providing the initial function for getting bids, ChatGPT suggested a modification to address indexing errors:

```
def get_bids(players):
    bids = {}
    for player in players:
        print(f"{player['name']}, choose a card to bid: ")
        print(f"Your cards: {player['cards']}")
        bid = input("Enter the rank of the card: ").upper()
        while bid not in RANKS or bid not in player['cards']:
            print("Invalid bid! Please choose a valid card from your hand.")
            bid = input("Enter the rank of the card: ").upper()
        bids[player['name']] = bid
    return bids
```

After modifications, the function works correctly:

```
def get_bids(players):
    bids = {}
    for player in players:
        print(f"{player['name']}, choose a card to bid: ")
        print(f"Your cards: {[(i+1, card) for i, card in enumerate(player['cards'])]}")
        # Print cards with index
        while True:
            bid_index = input("Enter the index of the card you want to bid (1-{}): "
                              .format(len(player['cards']))).strip()
            if bid_index.isdigit() and 1 <= int(bid_index) <= len(player['cards']):
                bid_index = int(bid_index)
                break
            else:
                print("Invalid input! Please enter a valid index.")
        bid_card = player['cards'].pop(bid_index - 1) # Remove card from player's hand
        bids[player['name']] = bid_card[0] # Add rank of the card to bids
    return bids
```

This modified function correctly handles indexing and allows players to bid effectively.

6 Transcript of Conversation with ChatGPT

The conversation with ChatGPT can be found in the following link: `https://docs.google.com/document/d/1HjM6X0OtmlAMFVMUWqX212qcl0wid3jswoo1Ukl_n40/edit?usp=sharing`

7 Final Code Implementation

The final code implementation can be found in the following link: `https://colab.research.google.com/drive/1LNZNbFdeBTHT_TldGhv8hZtG5qCda4MF?usp=sharing`

8 Analysis and Conclusion

- After a few tries, ChatGPT successfully provided me with the code for the "Diamonds" game. Initially, there were some mistakes in the code, but after making a few corrections, ChatGPT gave us a version that worked well. This process demonstrated how ChatGPT can learn and improve its solutions over time.

- The final code generated by ChatGPT has been included in the Colab file for reference. This demonstrates the usefulness of ChatGPT in solving complex problems and producing functional code.

- However, one drawback is that ChatGPT may not always provide the correct code in one go. It often requires multiple iterations and corrections to arrive at a functional solution. This limitation should be considered when using ChatGPT for code generation tasks.

- In conclusion, the process of refining the code with ChatGPT has highlighted its ability to learn and adapt to solve problems. This capability has significant potential in various software development tasks, offering an efficient method for creating working solutions.