

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

Khyati Satija

March 26, 2024

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Problem Statement</b>	<b>3</b>
<b>3</b>	<b>Teaching GenAI the game</b>	<b>4</b>
3.1	Rules of the Game . . . . .	4
3.2	The prompts given . . . . .	5
<b>4</b>	<b>Iterating upon the strategy</b>	<b>6</b>
4.1	Code for the strategy . . . . .	6
4.2	Results of playing against genAI . . . . .	9
<b>5</b>	<b>Analysis and Conclusion</b>	<b>11</b>

# Chapter 1

## Introduction

In this report , I will be prompting genAI to create a computer program which uses some optimizing strategy to play the game of Diamonds against me. I will be discussing and setting rules of the game and then ask chatGPT to strategically paly the game of Diamonds against me.

## Chapter 2

# Problem Statement

The problem is to prompt genAI to create a computer program that uses some optimizing strategy to play the game of Diamonds against the user.

## Chapter 3

# Teaching GenAI the game

While I was teaching genAI the game, it kept on discussing the 4 player diamonds game, So the first step for me was to make it play diamonds in a two player setting.

### 3.1 Rules of the Game

The objective remains the same: score points by collecting diamonds in tricks and using special cards strategically.

Players: Just you and me! We'll each play individually, and there's no need for partnerships.

Cards: We'll still use a standard deck of 52 playing cards. The cards rank from Ace (highest) down to 2 (lowest) in each suit.

Setup: 1. Shuffle the deck and deal out all the cards evenly between us. Each of us will have a hand of 26 cards.

2. We'll take turns being the dealer for each hand.

Gameplay: 1. Trick-taking: We'll still play a series of tricks. Each trick consists of one card played by each player.

2. Diamonds as Trump: As in the standard game, diamonds are trump. If a diamond card is led, the other player must play a diamond if they have one. If not, they can play any card.

3. Scoring: We'll score points in the same way as in the standard game: - Diamonds in Tricks: At the end of each trick, if there are any diamond cards in the trick, the player who wins the trick collects those diamonds and places them in their scoring pile. - Special Cards: We'll retain the special abilities of certain cards: - Diamonds (Trump) Cards: These cards not only win the trick but also allow the player to collect the diamonds from that trick. - Diamonds (Non-Trump) Cards: When played to a trick where diamonds are not led, and the opponent cannot follow suit, these cards count as one point each at the end of the hand. - Hearts Cards: These cards allow a player to "bury" diamonds from their hand, moving them from their hand to their scoring pile without playing them

to a trick. - Queen of Spades: This card allows a player to "catch" diamonds from their scoring pile and move them to their opponent's scoring pile.

4. End of Hand: The hand ends when all 26 cards have been played.

5. Scoring at the End of the Hand: Players count the number of diamonds they collected in tricks, the number of diamonds they buried using Hearts cards, and the number of diamonds they captured with the Queen of Spades. Each captured diamond is worth one point. Additionally, players score one point for each diamond card they have collected face down during the hand.

6. End of Game: The game can continue for a set number of hands, or until a predetermined score limit is reached. The player with the highest score at the end of the game wins!

Special Notes: - With only two players, the dynamic of the game will change significantly. Players will need to adjust their strategies accordingly, considering the absence of partnerships and the reduced number of cards in play.

## 3.2 The prompts given

I gave the following prompts to teach genAI the game of diamonds. - do you know how to play the game of diamonds? Chat gpt gave me the rules for a four player game and then i asked it to modify the rules for a two player game where them and I will be playing against each other.

Then, once we reached a consensus on the rules, I asked it to give me the code to play Diamonds against it using Python?

- can you give me the code for this game in python, where you and me play against each other?

## Chapter 4

# Iterating upon the strategy

Here are the ways I asked chat GPT to improvise the code and make the computer win :

**Added choose computer card Function:** Implemented a new function to determine the best card for the computer to play based on various factors, such as the presence of trump cards, the current trick, and the trump suit.

**Trump Management:** Prioritized playing high-value trump cards to win tricks and strategically chose between playing high-ranking and low-ranking trump cards based on the current trick.

**Leading Strategies:** Improved the decision-making process for the computer player when leading tricks, considering factors such as the potential to force the opponent to play valuable trump cards.

**Defensive Play:** Implemented defensive strategies to minimize the number of diamonds lost when the opponent is likely to win a trick.

**Burying Diamonds:** Incorporated logic for the computer player to strategically use Hearts cards to bury diamonds, especially when the computer has a lead in points.

**Adaptation to Opponent's Strategy:** Although not explicitly implemented in the code, the computer player's decision-making process is designed to adapt to the opponent's gameplay tendencies, such as adjusting its strategy based on the opponent's moves.

**Code Organization:** Organized the code into functions for better readability and maintainability, making it easier to understand and modify individual components.

### 4.1 Code for the strategy

```
import random

# Define global variables
```

```

SUITS = ['Hearts', 'Diamonds', 'Clubs', 'Spades']
RANKS = ['2', '3', '4', '5', '6', '7', '8', '9', '10',
'Jack', 'Queen', 'King', 'Ace']
POINTS_TO_WIN = 100 # Adjust as needed

# Initialize player scores
your_score = 0
computer_score = 0

# Function to create and shuffle the deck
def create_deck():
    deck = [(rank, suit) for rank in RANKS for suit in SUITS]
    random.shuffle(deck)
    return deck

# Function to deal hands to players
def deal_hands(deck):
    return deck[:26], deck[26:]

# Function to determine if a card is a trump card
def is_trump(card):
    return card[1] == 'Diamonds'

# Function to determine the winning card in a trick
def winning_card(trick, trump_suit):
    trick.sort(key=lambda x: (x[1] != trump_suit, RANKS.index(x[0])))
    return trick[-1]

# Function to play a trick
def play_trick(your_hand, computer_hand, trump_suit):
    trick = []
    if random.random() < 0.5: # Computer plays first 50% of the time
        card = computer_hand.pop(0)
        print("Computer plays", card)
        trick.append(card)
    while True:
        print("Your hand:", your_hand)
        if not your_hand:
            return trick
        print("Choose a card to play:")
        for i, card in enumerate(your_hand):
            print(i + 1, ":", card)
        choice = input("Enter the number of the card you want to play, or 0 to pass")
        if choice == '0':
            return trick
        try:
            choice = int(choice)
            if 1 <= choice <= len(your_hand):
                card = your_hand.pop(choice - 1)
                print("You play", card)

```



```

        trick.append(card)
        break
    else:
        print("Invalid choice. Please enter a number between 1 and", len(your_cards))
except ValueError:
    print("Invalid input. Please enter a number.")

# Function to determine the best card for the
# computer to play
def choose_computer_card(computer_hand, trick, trump_suit):
    if not trick:
        return computer_hand[0]

    # Check if the computer has any trump cards left
    has_trump = any(is_trump(card) for card in computer_hand)

    # Check if the computer should play
    # a high-ranking trump card to win the trick
    if has_trump and trick[0][1] == trump_suit:
        high_trump_cards = [card for card in computer_hand
                             if is_trump(card) and RANKS.index(card[0]) > RANKS.index(trick[-1][0])]
        if high_trump_cards:
            return max(high_trump_cards, key=lambda
                        x: RANKS.index(x[0]))

    # Check if the computer should play a
    # low-ranking trump card to avoid
    # wasting high-ranking cards
    if has_trump:
        low_trump_cards = [card for card in computer_hand if is_trump(card) and RANKS.index(card[0]) < RANKS.index(trick[-1][0])]
        if low_trump_cards:
            return min(low_trump_cards, key=lambda x: RANKS.index(x[0]))

    # If there are no trump cards left or it's not advantageous
    # to play a trump card,
    # play a non-trump card
    non_trump_cards = [card for card in computer_hand if not is_trump(card)]
    if non_trump_cards:
        return max(non_trump_cards, key=lambda x: RANKS.index(x[0]))

    # If the computer has no choice but to play a trump card,
    # play the highest-ranking one
    return max(computer_hand, key=lambda x: RANKS.index(x[0]))

# Main game loop
while True:
    print("\nYour score:", your_score)
    print("Computer's score:", computer_score)

```

```

# Create and shuffle the deck
deck = create_deck()

# Deal hands
your_hand, computer_hand = deal_hands(deck)

# Determine trump suit
trump_suit = 'Diamonds'

# Play tricks until both players run out of cards
while your_hand or computer_hand:
    trick = play_trick(your_hand, computer_hand, trump_suit)
    winning_card = winning_card(trick, trump_suit)
    if is_trump(winning_card):
        if trick.index(winning_card) % 2 == 0:
            your_score += len(trick)
        else:
            computer_score += len(trick)
    else:
        if trick.index(winning_card) % 2 == 0:
            your_score += sum(1 for card in trick if is_trump(card))
        else:
            computer_score += sum(1 for card in trick if is_trump(card))
    # Computer plays
    computer_card = choose_computer_card(computer_hand, trick, trump_suit)
    print("Computer plays", computer_card)
    trick.append(computer_card)
    computer_hand.remove(computer_card)

# Check if a player has won
if your_score >= POINTS_TO_WIN or computer_score >= POINTS_TO_WIN:
    print("\nFinal Scores:")
    print("Your score:", your_score)
    print("Computer's score:", computer_score)
    if your_score >= POINTS_TO_WIN:
        print("Congratulations! You win!")
    else:
        print("Sorry, the computer wins.")
    break

# Ask to play another round
play_again = input("Do you want to play another round? (yes/no): ").lower()
if play_again != 'yes':
    break

```

## 4.2 Results of playing against genAI

There were instances of me winning and sometimes genAI also won. It was overall an interesting and learning experience by teaching genAI to play the

game of diamonds against me in a strategically manner.

## Chapter 5

# Analysis and Conclusion

Game Overview: ChatGPT provided an explanation of the basic rules and objectives of the Diamonds card game, including how players score points by collecting diamonds in tricks and using special cards strategically.

Two-Player Adaptation: adapted the rules of Diamonds for two players, allowing computer and user to play against each other. modified the gameplay and scoring rules accordingly.

Python Implementation: chatGPT provided a Python code implementation of the Diamonds card game for two players, where user can play against the computer. The code includes functions for dealing cards, playing tricks, and determining the winner.

Strategies for Computer Player: discussed various strategies to enhance the computer player's performance in the game, such as trump management, leading strategies, defensive play, and adaptation to the opponent's strategy. These strategies were then incorporated into the code to improve the computer player's decision-making process.

Code Improvements: improvements were made to the code to enhance the computer player's performance, including the addition of a function to choose the best card to play, implementation of strategic decision-making, and organization of the code for readability and maintainability.