Teaching genAI to Play Diamonds: An AI Learning Journey

March 26, 2024

Introduction/Objective/Problem Statement

The objective of this report is to document the process of teaching an AI, to play the game of Diamonds using an optimizing strategy. The game of Diamonds involves bidding with cards to acquire valuable diamond cards, ultimately aiming to accumulate the most points to win. The challenge lies in developing a strategy that optimizes bidding decisions to maximize point acquisition. By prompting genAI to create a computer program and teaching it the rules and strategies of the game, we aim to explore how AI can learn and adapt to complex gaming scenarios.

Methodology

The process began by outlining the rules of the Diamonds game, including card distribution, auction phases, bidding, and scoring. Subsequently, genAI was prompted with instructions to create a computer program capable of playing Diamonds optimally. The prompts provided to genAI included detailed explanations of the game rules, along with strategies for effective bidding and point accumulation.

Reflections on Conversation with genAI & Prompting/Learnings

Engaging with genAI to teach it the game of Diamonds was an enlightening experience. Through structured prompts, genAI demonstrated a remarkable ability to grasp the complexities of the game and generate code accordingly. The conversations with genAI provided insights into various bidding strategies, including considerations for card valuation, risk assessment, and adaptive decision-making. Moreover, the process highlighted the importance of clear and concise communication in guiding AI learning.

Reflections on Code that was Generated/Snippets

The code generated by genAI exhibited a logical approach to playing the game of Diamonds. It incorporated elements of strategic bidding, including evaluating card values, assessing opponents' bids, and optimizing point acquisition. Snippets of the generated code showcased algorithms for determining bid amounts based on the current state of the game and the potential value of diamond cards.

Conclusion & Path Forward

In conclusion, teaching genAI to play the game of Diamonds demonstrated the potential of AI in mastering complex gaming scenarios. The process highlighted the effectiveness of structured prompts in guiding AI learning and the importance of strategic thinking in game optimization. Moving forward, further refinement of the AI's bidding strategy and extensive testing against human players could enhance its gameplay capabilities. Additionally, exploring advanced AI techniques such as reinforcement learning may offer insights into evolving dynamic strategies for optimal performance in the Diamonds game.

Appendices

GenAI Transcripts:

ChatGPT Transcripts

Code:

```
import random

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

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

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

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

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

    def get_highest_card(self):
        return max(self.hand, key=lambda card: card.rank_value)

class Game:
    def __init__(self, players):
        self.players = players
        self.diamond_cards = []
        self.suits = ['Hearts', 'Spades', 'Clubs']
        self.create_deck()
        self.deal_cards()

    def create_deck(self):
        ranks = {'2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7, '8': 8, '9': 9, '10': 10, 
        for suit in self.suits:
            for rank, value in ranks.items():
                self.diamond_cards.append(Card(rank, suit, value))
        random.shuffle(self.diamond_cards)

    def deal_cards(self):
        for player in self.players:
            for _ in range(3):  # Deal 3 cards to each player
                player.add_card_to_hand(self.diamond_cards.pop())

    def auction(self):
        for diamond_card in self.diamond_cards:
            bids = {}
            for player in self.players:
                # Simulate player bidding, assuming player selects a random card from their
                bid_card = random.choice(player.hand)
                bids[player] = bid_card

            highest_bid = max(bids.values(), key=lambda card: card.rank_value)
            winners = [player for player, card in bids.items() if card.rank_value == highest

            if len(winners) == 1:
                winner = winners[0]
                winner.remove_card_from_hand(bids[winner])
                winner.add_card_to_hand(diamond_card)
                # Add points to winner's score
                winner.score += diamond_card.rank_value
            else:
                points_per_winner = diamond_card.rank_value / len(winners)
                for winner in winners:
                    winner.remove_card_from_hand(bids[winner])
                    winner.add_card_to_hand(diamond_card)
                    # Add points to winner’s score
                    winner.score += points_per_winner

    def display_scores(self):
        print("Scores:")
        for player in self.players:
            print(f"{player.name}: {player.score}")

    def play(self):
        self.auction()
        self.display_scores()

if __name__ == "__main__":
    # Example usage
    player1 = Player("Alice")
    player2 = Player("Bob")
    players = [player1, player2]
    game = Game(players)
    game.play()
```