Diamond Game Report

Aastha Deshwal

March 26, 2024

Abstract

This report details the interaction with a generative AI (genAI) called Gemini to develop a strategy for the trick-taking card game "Diamonds." The objective was to create an optimized Diamonds game simulation code in Python, which reduces the chance of losing the game.

0.1 Introduction/Objective

The below report outlines the interaction with the generative AI (genAI) called Gemini, which is put to use while developing a strategy for the trick-taking card game "Diamonds." The objective was to generate an optimized Diamonds game simulation code in Python, which reduces the chance of losing the game.

0.2 Methodology

The development process involved the following steps:

1. Define Game Rules: We began by establishing a clear understanding of the Diamonds game rules, including bidding, playing tricks, and scoring. This ensured a common platform among humans and AI to share the concept.

2. Develop Strategies: We discussed potential strategies for bidding and card selection with Gemini. The primary focus was on minimizing the possibility of losing hands, particularly in scenarios where the risk of losing is not necessarily high. A crucial piece of information would be shared about possible strategies; Gemini would be able to create, while we will continue to prompt him to generate an optimal code for the Diamonds simulation game.

3. Code Generation Optimization: Once we had the strategy, the code was generated by Gemini in Python code to simulate the Diamonds game. Further refinements were carried out in the code for better readability and optimality. After this step, we discussed some test cases in order to simulate the Diamonds game and check the simulation code.

0.3 Reflection on Conversation with genAI Learnings

Communicating with Gemini was fruitful and afforded an opportunity to refine the approach in developing the Diamonds game simulation code. Below are some of the highlights that I experienced during our interaction:
* Importance of Clear Communication: A clear definition of the game rules ensured the AI concentrated on its mission.
* Exploring Different Strategies: Discussing different strategies with Gemini opened our minds to other strategies.
* Leveraging AI Capabilities: This capability of Gemini in generating and optimizing the code significantly sped up development.

0.4 Reflections on Code That Was Generated/Snippets

The diamonds-game code is well-structured in terms of the simulation process. Here are some of the key points regarding the code:

* Card and Suit Definitions: Defining card suits and ranks with brief formulations makes the code readable.
* Card Comparison Overloading: Overloading the "¿" operator makes card comparison in terms of rank and trump easier during trick evaluation.
* Bidding Strategy: The bidding strategy considers high cards and balanced suits, leading to more strategic bids.
* Playing Cards Strategy: The card selection prioritizes Diamonds and high cards within playable suits, enhancing the strategic depth of gameplay.

Snippet code below for the optimized comparison of cards:

python
import random

Card and Suit definitions (more concise)
SUITS = "hearts diamonds clubs spades".split()
RANKS = "A K Q J 10 9 8 7 6 5 4 3 2".split()

class Card:
def ${}_{init}{}_{(self,suit,rank):self.suit=suitself.rank=rank}$
def ${}_{gt}{}_{(self,other):Overloading">"operatorforcardcomparisonbasedonrankandtrumpsuit_values="hearts":0,"diamonds":1,"clubs":2,"spa}$
class Deck:
def ${}_{init}{}_{(self):self.cards=[Card(suit,rank)forsuitinSUITSforrankinRANKS]}$
def shuffle(self):
random.shuffle(self.cards)

def deal(self, num$_players, cards_per_player)$:
$hands = [[]for_inrange(num_players)]$
$for_inrange(cards_per_player)$:
$forplayerinrange(num_players)$:
$hands[player].append(self.cards.pop())$
$returnhands$

def get$_bid(hand)$:
$Improvedbiddingstrategy : considershighcardsandbalancedsuits$
$high_cards = sum(rankin("A","K","Q","J")forcardinhand)$
$suit_counts = suit : 0forsuitinSUITS$
$forcardinhand$:
$suit_counts[card.suit]+ = 1$
$balanced_suit_bonus = sum(count >= 3forcountinsuit_counts.values())$
$returnmin(high_cards + balanced_suit_bonus, 7)$

def play$_card(hand, led_suit)$:
$Morestrategiccardselection : prioritizeDiamondsandhighcards$
$best_card = max($

$card$
$forcardinhand$
$if(card.suit == led_suitorcard.suit == "diamonds")$
$andcardnotinhand$
$)$
$ifnotbest_card : NoplayablesuitorDiamonds, playthehighestcard$
$best_card = max(hand)$
$returnbest_card$

def evaluate$_trick(played_cards) :$
$Usemaxfunctionwithcustomcomparisonforwinnerdetermination$
$returnmax(played_cards)$

def calculate$_points(bid, tricks_won) :$
$returnbid * 10 + max(0, tricks_won - bid)$

def play$_game(num_players) :$
$deck = Deck()$
$deck.shuffle()$
$hands = deck.deal(num_players, 13)$

Get bids from each player
bids = [get$_bid(hand)forhandinhands$]

Play each trick
tricks = []
for $_inrange(13) :$
$led_suit = hands[0][0].suit$
$played_cards = [player.hand.pop(0)forplayerinhands]$
$winner = evaluate_trick(played_cards)$
$tricks.append((led_suit, winner.suit, winner.rank))$

Reorder players based on who won the trick
winning$_index = played_cards.index(winner)$
$hands = hands[winning_index :] + hands[: winning_index]$

Calculate points for each player
for i, hand in enumerate(hands):
tricks$_won = sum(trick[1] == bids[i]fortrickintricks)$
$points = calculate_points(bids[i], tricks_won)$
$print(f"Playeri + 1Points : points")$

0.5 Conclusion Path Forward

Developing a functional and strategic Diamonds game simulation code with the assistance of Gemini worked. Further dimensions of improvement that we would like to aim at are:

Advanced Strategies: The creation of more complicated strategies, like underbidding or making sacrifices to win the later game. AI Opponent: Creating an AI opponent of varying levels of skill with which to test the effectiveness of the code with regard to various levels of playing. GUI: The creation of a graphical user interface for the Diamonds game simulation, allowing the players to have a direct view and interaction with the code. These improvements should provide a much more robust and flexible tool that can be used to make any sort of Diamonds simulation.