"Enhancing the Diamonds game: Computer Strategy and UI Development with Pygame using GenAI"

Atchaya M, Shivani S

April 8, 2024

1 Introduction

This report is based on my experience of teaching Gemini AI a card game and evaluating how well it understood the rules and strategies. As a person who knows the rules of the game, I took on the role of teaching Gemini AI the ins and outs of the game. Our conversation had its ups and downs, with Gemini AI initially making some assumptions and getting a few things wrong. However, with explanation and more accurate prompts, Gemini AI eventually grasped the game's mechanics and even showed interest in coming up with strategies to win.

2 Teaching GenAI the game

When I started explaining the game to Gemini AI, it had a few misconceptions. It made some assumptions that weren't quite right. But I corrected those assumptions and walked through the rules again. It took a bit of back and forth, but eventually, Gemini AI got the hang of it. It was eager to learn and improve its understanding. We went over the rules multiple times, clarifying any points of confusion until Gemini AI had a clear picture of how the game worked. At first, its responses were brief summaries, and it even had some hallucinations, but with more explanation, it started to grasp the game better.

3 Iterating upon strategy

As our conversation progressed, Gemini AI expressed interest in figuring out strategies to win the game. Interestingly, our conversation didn't touch on a specific strategy my peers shared in class, called the threshold value strategy. Instead the strategies used were:

1) Prioritize High Value Diamonds: Gemini AI suggested focusing on winning high-value diamonds (Jacks, Queens, Kings, and Aces) to maximize point accumulation.

2) Assess Hand Strength: It emphasized evaluating hand strength before bidding, distinguishing between low-value and high-value cards within its own suit.

3) Bidding Strategy based on Revealed Diamonds: Gemini AI recommended adjusting bidding tactics based on diamonds already revealed, prioritizing lower-value cards for remaining diamonds when high-value ones are claimed.

4) Balancing Bidding and Suit Strength: It acknowledged the importance of balancing bidding and suit strength to remain competitive in both bidding and trick-taking aspects of the game.

5) Limited Bluffing: Gemini AI incorporated limited bluffing into its gameplay, occasionally using low-value cards to bid on high-value diamonds to deceive opponents.

4 Analysis and Conclusion

While my conversation with Gemini AI yielded some valuable insights and strategies for the card game, it's worth noting that it didn't reach the same level of depth as some of my peers' interactions. In their conversations, Gemini AI provided excellent strategies by dividing the game into early, mid, and late rounds, along with offering general strategies. Additionally, when asked to write code, it produced code based on the strategies discused above instead of the strategy of calculating a threshold value that is given by the formula:

min bid value = (revealed diamond + 1) / (num players * (14- round number))

that were presented by my peers in class. But, my conversation with Gemini AI didn't explore these advanced strategies or involve the implementation of complex code logic.

5 Final code

```
import pygame
import os
import random

# Initialize Pygame
pygame.init()

# Set up the display
SCREEN_WIDTH = 800
SCREEN_HEIGHT = 600
screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
pygame.display.set_caption("Card␣Game")
```

```
# Colors
GREEN = (0, 128, 0)  # Green color for background
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)

# Load card images
card_dir = "D:\Projects\Card game\PNG-cards-1.3\PNG-cards-1.3"
card_back_image = pygame.image.load(os.path.join(card_dir, "
    card_back.png"))  # Card back image
# Set the size for all cards to be displayed
card_width = 60
card_height = 92
card_padding = 1  # Adjust the padding between cards

# Start Button
start_button_font = pygame.font.Font(None, 36)
start_button_text = start_button_font.render("START GAME", True,
    BLACK)
start_button_rect = start_button_text.get_rect(center=(
    SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2))

# Confirm Button
confirm_button_font = pygame.font.Font(None, 24)
confirm_button_text = confirm_button_font.render("CONFIRM", True,
    BLACK)
confirm_button_rect = confirm_button_text.get_rect(center=(
    SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2 + 150))

# Close Button
close_button_font = pygame.font.Font(None, 36)  # Increase font
    size for the close button text
close_button_text = close_button_font.render("CLOSE", True, BLACK)
close_button_width, close_button_height = close_button_text.
    get_size()
close_button_rect = pygame.Rect((SCREEN_WIDTH - close_button_width
    ) // 2, SCREEN_HEIGHT // 2 + 250, close_button_width,
    close_button_height)  # Create a rectangle with the size of the
     text

# Function to initialize the deck
def initialize_deck():
    suits = ['Hearts', 'Clubs', 'Spades']
    ranks = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'Jack',
        'Queen', 'King', 'Ace']
    deck = [(rank, suit) for suit in suits for rank in ranks]
    random.shuffle(deck)
    return deck

# Function to deal cards to players
def deal_cards(deck):
    players = {
        'Player': [],
        'Computer': []
    }
    for _ in range(13):
        players['Player'].append(deck.pop())
        players['Computer'].append(deck.pop())
    return players
```

```
# Function to determine rank value
def get_rank_value(rank):
    rank_values = {
        '2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7, '8': 8, '9
           ': 9, '10': 10,
        'Jack': 11, 'Queen': 12, 'King': 13, 'Ace': 14
    }
    return rank_values.get(rank, 0)

# Function to determine computer bid based on strategy
def get_computer_bid(diamond, computer_hand, used_cards,
   num_cards_remaining, computer_used_cards):
    diamond_value = get_rank_value(diamond[0])
    THRESHOLD = 4

    if diamond_value >= 10 or (num_cards_remaining <= THRESHOLD
       and len(computer_hand) > 1):
        for card in computer_hand:
            if card not in used_cards and card not in
               computer_used_cards and get_rank_value(card[0]) >=
               8:
                return card
    elif diamond_value < 8:
        for card in computer_hand:
            if card not in used_cards and card not in
               computer_used_cards and get_rank_value(card[0]) <=
               6:
                return card

    available_cards = [card for card in computer_hand if card not
       in used_cards and card not in computer_used_cards]
    return random.choice(available_cards)

# Function to conduct auction and determine winner
def conduct_auction(player_card, computer_card, center_card):
    player_rank = get_rank_value(player_card[0])
    computer_rank = get_rank_value(computer_card[0])
    center_rank = get_rank_value(center_card[0])
    if player_rank > computer_rank:
        return 'player'
    elif player_rank < computer_rank:
        return 'computer'
    else:
            return 'tie'

# Function to update scores
def update_scores(result, center_card_rank):
    global player_score, computer_score
    if result == 'player':
        player_score += center_card_rank
    elif result == 'computer':
        computer_score += center_card_rank
    else:
        player_score += center_card_rank // 2
        computer_score += center_card_rank // 2

# Function to display rules
```

```
def display_rules():
    rules_font = pygame.font.Font(None, 24)
    rules_text = [
        "RULES:",
        "1. Each player gets a suit of cards other than the
            diamond suit.",
        "2. The diamond cards are then shuffled and put on auction
            one by one.",
        "3. All the players must bid with one of their own cards
            face down.",
        "4. The banker gives the diamond card to the highest bid,
            i.e. the bid with the most points.",
        "    2<3<4<5<6<7<8<9<T<J<Q<K<A",
        "5. The winning player gets the points of the diamond card
            to their column in the table.",
        "    If there are multiple players that have the highest
            bid with the same card,",
        "    the points from the diamond card are divided equally
            among them.",
        "6. The player with the most points wins at the end of the
            game."
    ]
    for i, line in enumerate(rules_text):
        text = rules_font.render(line, True, BLACK)
        screen.blit(text, (20, 20 + i * 20))  # Adjust the
            position as needed

# Set up game variables
game_started = False
center_card = None
confirm_clicked = False
selected_player_card = None
selected_player_card_index = None
selected_computer_card_index = None
player_score = 0
computer_score = 0
round_count = 0
diamond_cards = [(rank, 'Diamonds') for rank in ['2', '3', '4', '5
    ', '6', '7', '8', '9', '10', 'Jack', 'Queen', 'King', 'Ace']]

# Adjust display positions for scorecard
scorecard_height = 150  # Height of the scorecard area
card_display_height = SCREEN_HEIGHT - scorecard_height + 150  #
    Adjusted height for card display area

# Game Loop
running = True
while running:
    screen.fill(GREEN)  # Set background color to green

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        if event.type == pygame.MOUSEBUTTONDOWN and not
            game_started:
            if start_button_rect.collidepoint(event.pos):
                game_started = True
                # Initialize the deck
```

```
            display_rules()
            pygame.display.flip()
            pygame.time.delay(10000)
            deck = initialize_deck()
            # Deal cards to players
            players = deal_cards(deck)
            player_cards = players['Player']
            computer_cards = players['Computer']
            # Select a random card from the diamond suit for
               the center card
            center_card = random.choice(diamond_cards)
            diamond_cards.remove(center_card)
    elif event.type == pygame.MOUSEBUTTONDOWN and game_started
        :
        if round_count < 13:
            if confirm_button_rect.collidepoint(event.pos):
                if selected_player_card is not None:
                    # Determine computer bid based on strategy
                    computer_bid = get_computer_bid(
                        center_card, computer_cards, [], len(
                        deck), [])
                    selected_computer_card_index =
                        computer_cards.index(computer_bid)
                    confirm_clicked = True
                    # Conduct auction and update scores
                    result = conduct_auction(
                        selected_player_card, computer_bid,
                        center_card)

                    # Display the selected computer card face-
                        up
                    selected_computer_card = computer_cards[
                        selected_computer_card_index]
                    selected_card_image = pygame.transform.
                        scale(pygame.image.load(os.path.join(
                        card_dir, f"{selected_computer_card[0]}
                        _of_{selected_computer_card[1].lower()
                        }.png")), (card_width, card_height))
                    selected_card_rect = selected_card_image.
                        get_rect(center=(SCREEN_WIDTH // 2 -
                        100, SCREEN_HEIGHT // 2))
                    screen.blit(selected_card_image,
                        selected_card_rect)

                    # Display the selected player card face-up
                    selected_player_card_image = pygame.
                        transform.scale(pygame.image.load(os.
                        path.join(card_dir, f"{
                        selected_player_card[0]}_of_{
                        selected_player_card[1].lower()}.png"))
                        , (card_width, card_height))
                    selected_player_card_rect =
                        selected_player_card_image.get_rect(
                        center=(SCREEN_WIDTH // 2 + 100,
                        SCREEN_HEIGHT // 2))
                    screen.blit(selected_player_card_image,
                        selected_player_card_rect)
```

```
                    # Update the display
                    pygame.display.flip()

                    # Delay before updating scores
                    pygame.time.wait(1500)

                    # Update scores
                    update_scores(result, get_rank_value(
                        center_card[0]))

                    pygame.time.wait(250)

                    # Remove selected cards from players'
                        hands
                    player_cards.remove(selected_player_card)
                    computer_cards.remove(
                        selected_computer_card)

                    round_count += 1
                    if round_count < 13:
                        # Select a new random card from the
                            diamond suit for the center card
                        center_card = random.choice(
                            diamond_cards)
                        diamond_cards.remove(center_card)
                    # Reset selection variables
                    selected_player_card = None
                    selected_player_card_index = None
                    selected_computer_card_index = None
                    confirm_clicked = False
            else:
                # Game over
                running = False

            # Check if any player card is clicked
            for i, (rank, suit) in enumerate(player_cards):
                card_rect = pygame.Rect((card_width + card_padding
                    ) * i + (SCREEN_WIDTH - (card_width +
                    card_padding) * len(player_cards)) // 2,
                    card_display_height - card_height - 20,
                    card_width, card_height)
                if card_rect.collidepoint(event.pos):
                    if selected_player_card_index == i:  # If
                        already selected, deselect
                        selected_player_card_index = None
                        selected_player_card = None
                    else:  # Select new card
                        selected_player_card_index = i
                        selected_player_card = (rank, suit)

    if game_started:
        if round_count < 13:
            # Display computer's cards
            total_computer_width = (card_width + card_padding) *
                len(computer_cards)  # Calculate the total width of
                 all computer cards
            start_computer_x = (SCREEN_WIDTH -
                total_computer_width) // 2
```

```
            start_computer_y = 180  # Adjusted starting Y-
                coordinate for computer cards
            for i, (rank, suit) in enumerate(computer_cards):
                card_image = pygame.transform.scale(
                    card_back_image, (card_width, card_height))  #
                    Scale the card back image to match player cards
                screen.blit(card_image, (start_computer_x,
                    start_computer_y))
                start_computer_x += card_width + card_padding  #
                    Adjusted spacing between computer cards

            # Display center card
            # Calculate the Y-coordinate for the center diamond
                card
            diamond_y = int(SCREEN_HEIGHT * 0.62)  # 38% from the
                bottom

            # Load and scale the center diamond card image
            center_card_image = pygame.transform.scale(pygame.
                image.load(os.path.join(card_dir, f"{center_card
                [0]}_of_{center_card[1].lower()}.png")), (
                card_width, card_height))

            # Create the rectangle for the center diamond card
            center_card_rect = center_card_image.get_rect(center=(
                SCREEN_WIDTH // 2, diamond_y))

            # Blit the center diamond card onto the screen
            screen.blit(center_card_image, center_card_rect)

            # Display player's cards at the bottom of the screen
            total_player_width = (card_width + card_padding) * len
                (player_cards)
            start_player_x = (SCREEN_WIDTH - total_player_width)
                // 2
            for i, (rank, suit) in enumerate(player_cards):
                card_image = pygame.transform.scale(pygame.image.
                    load(os.path.join(card_dir, f"{rank}_of_{suit.
                    lower()}.png")), (card_width, card_height))
                if selected_player_card_index == i:
                    screen.blit(card_image, (start_player_x,
                        card_display_height - card_height - 40))
                else:
                    screen.blit(card_image, (start_player_x,
                        card_display_height - card_height - 20))
                start_player_x += card_width + card_padding  #
                    Adjusted spacing between player cards

            # Display confirm button if a player card is selected
            if selected_player_card is not None:
                pygame.draw.rect(screen, (200, 200, 200),
                    confirm_button_rect)
                screen.blit(confirm_button_text,
                    confirm_button_rect.topleft)
        else:
            # Game over, display final scores
            font = pygame.font.Font(None, 48)
            player_score_text = font.render(f"Your␣final␣score:␣{
```

```
            player_score} points", True, BLACK)
        computer_score_text = font.render(f"Computer's final
            score: {computer_score} points", True, BLACK)
        screen.blit(player_score_text, (SCREEN_WIDTH // 2 -
            player_score_text.get_width() // 2, SCREEN_HEIGHT
            // 2 - 50))
        screen.blit(computer_score_text, (SCREEN_WIDTH // 2 -
            computer_score_text.get_width() // 2, SCREEN_HEIGHT
             // 2 + 50))
        if player_score > computer_score:
            winner_text = "Congratulations, You win!!"
        elif player_score < computer_score:
            winner_text = "The computer wins!!"
        else:
            winner_text = "That's a tie!!"
        winner = font.render(winner_text, True, BLACK)
        screen.blit(winner, (SCREEN_WIDTH // 2 - winner.
            get_width() // 2, SCREEN_HEIGHT // 2 + 150))

        # Display close button
        pygame.draw.rect(screen, (200, 200, 200),
            close_button_rect)
        screen.blit(close_button_text, close_button_rect.
            topleft)

else:
    # Display start button
    pygame.draw.rect(screen, (200, 200, 200),
        start_button_rect)
    screen.blit(start_button_text, start_button_rect.topleft)

# Draw scorecard
if game_started:
    # Create scorecard surface
    scorecard_surface = pygame.Surface((300, 100))
    scorecard_surface.fill(WHITE)
    scorecard_rect = scorecard_surface.get_rect(center=(
        SCREEN_WIDTH // 2, scorecard_height // 2))
    # Draw outline for scorecard
    pygame.draw.rect(scorecard_surface, BLACK,
        scorecard_surface.get_rect(), 2)
    # Render text on scorecard
    score_font = pygame.font.Font(None, 24)
    player_score_text = score_font.render(f"Your points: {
        player_score} points", True, BLACK)
    computer_score_text = score_font.render(f"Computer's
        points: {computer_score} points", True, BLACK)
    # Blit text onto scorecard surface
    scorecard_surface.blit(player_score_text, (20, 20))
    scorecard_surface.blit(computer_score_text, (20, 50))
    # Blit scorecard onto screen
    screen.blit(scorecard_surface, scorecard_rect)

pygame.display.flip()

# Event handling for the close button
if not running:
    if event.type == pygame.MOUSEBUTTONDOWN:
```

```
            if close_button_rect.collidepoint(event.pos):
                running = False

# Quit Pygame
pygame.quit()
```

Listing 1: Python Code for Game Strategy

6 Transcript

Gemini AI : https://g.co/gemini/share/8c18706dfc1b