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

Atchaya M, Shivani S

April 8, 2024

Git Repository Link

1 Introduction

In this card game, each player is assigned a suit of cards excluding the diamond suit. The diamond cards are shuffled and auctioned off individually, with all players required to bid using one of their own cards face down. The banker awards the diamond card to the highest bidder, who possesses the card with the highest point value based on the hierarchy from 2 to Ace. The winning player receives the points of the diamond card into their score, potentially sharing the points if multiple players tie for the highest bid with the same card. The ultimate objective is for a player to accumulate the most points by the end of the game to emerge victorious.

2 Methodology

Objective

Our objective was to familiarize a selected generational AI with the rules of the game and subsequently analyze winning strategies for players. Furthermore, we aimed to task the AI with generating the game execution code, along with a user interface using Pygame.

Familiarization with the rules

- We spent a significant amount of effort in understanding the game ourselves. We worked out various strategies that we thought would lead to victory. Additionally, we recognized that human psychology, including factors such as

facial expressions and conversational cues, could significantly influence gameplay. Consequently, our ultimate aim was to enable the computer to play the game with a strategic approach rather than through random decision-making.

- After this, each of us picked one AI (ChatGPT and Gemini for the two of us involved) and started teaching the rules to it. We asked it for various doubts and provided clarifications.

- Then we proceeded to play a game with it. This is when we realized that the AI still had trouble getting a lot of the rules down. We also realized that there was a gap in the information we provided and that we should have been more careful and detailed in providing it with information. Some of the mistakes it made include:

  - For ChatGPT:
    * Incorrectly believed it could play the same card multiple times.
    * Failed to distribute an equal number of cards to the computer and the human player.
    * Inadvertently revealed the computer's hand while playing.
    * Erroneously distributed diamond cards to both players instead of being only present in the auction.
    * Initially did not play thirteen rounds as per the game rules.
    * Made errors in score-keeping.
    * Allowed the same card to be present on both sides - human and computer.
    * Failed to remove cards after bidding and did not correctly replace the card with a diamond card as requested.
    * Was slightly biased. The AI acted as both the auctioneer and the computer player. Now, it made the choice to bid only after I did and hence always chose a card a rank above than my card.
  - For Gemini:
    * Did not provide the list of cards after distribution and simply asked the human player to choose at random.
    * Distributed cards with diamonds, despite it being explained in the rules not to include them.
    * Simulated a rule independently.
    * Initially offered generic strategies only and provided specific responses that could be coded only upon request.
    * Incorrectly assumed the presence of 3 suits of cards, despite there being only 2 players.
    * Did not provide the human player with the specified 5 options to choose from, instead directing them to make a choice.

- This detailed account showcases the specific challenges encountered during the initial phases of interaction with the AIs and highlights the need for

improved clarity and specificity in the instructions provided to the AIs. A general complaint with GenAI is that it had no memory. We had to repeatedly keep track of what was going on and inform it so that it could generate proper results.

Coding using Prompts

We proceeded with coding using the respective AI, initially focusing on the logic and code of the entire game before integrating a strategy for the computer to win.

Strategy

- **Threshold**: The function starts by setting a threshold value of 4, which is used to determine the number of remaining cards where a specific strategy applies.
- **High-Value Diamonds**: If the value of the diamond card is 10 or higher, or if the number of remaining cards is equal to or below the threshold and the computer has more than one card in hand, the computer will prioritize bidding with a card of a value 8 or higher (if available and not already used).
- **Low-Value Diamonds**: If the value of the diamond card is less than 8, the computer will prioritize bidding with a card of a value 6 or lower (if available and not already used).
- **Available Cards**: If none of the above conditions are met, the computer will randomly select a card from its hand that has not been used yet.

This process consumed a significant amount of time, and despite multiple attempts to convey the rules, the AI did not generate the right code for a prolonged period. To improve code generation, we found that initiating a fresh chat and providing the AI with the code from our latest checkpoint increased the likelihood of obtaining the correct code. Eventually, we achieved the final working code for the game.

We found that pointing out exactly where the problem lay, rather than asking the AI to find it independently, led to more successful results. Additionally, providing hints toward the right answer or methodology proved to be helpful in guiding the AI.

Next, we started working on the UI part of the game. This took us longer than just getting the code for the logic of the game. Additionally, we started utilizing multiple AI for code generation, including the new tool, Merlin AI. While our use of Merlin was not extensive, it proved beneficial in rectifying errors that ChatGPT or Gemini were not correcting promptly.

Our approach to coding the game was to proceed step-by-step instead of providing the AI with the code and requesting a UI. We believed that this approach

offered better chances of obtaining accurate results and made the process easier to navigate.

We commenced with the start game page before progressing to the cards display page. Although we initially encountered alignment issues, with the diamond cards being distributed randomly to both the player and the computer, we refined the choosing functionality, ultimately achieving the proper display of the cards. We then proceeded to conceal the computer's cards, implemented the choose card and confirmation options for the player, and allowed the computer to select its card using the bidding strategy. Subsequently, we incorporated the scorecard and gaming loop before adding the winner display and close button.

Alignment posed a significant challenge, but ultimately, we attained the desired results.

3 Testing and Iteration :

We tested the game multiple times by playing with it. It was also a fun experience playing with the computer. We lost to the computer about 3 times before winning our first game. We also had our family members play against it.

4 Overall experience

Reflecting on our experience, we felt that it was an interesting and relatively easy way to learn about the workings of GenAI. There were times when we wanted to manually intervene and start coding ourselves - because AI simply wasn't giving us the results we wanted - but we held back and asked it to generate the code. However, AI did serve as a wonderful tool in helping us code this game.

We gained a better perspective as to which prompts will provide better results. We also realized how detailed we had to be in giving instructions. Overall it was a wonderful assignment.

5 Diamond Card Game UI - Demo Video

Visit Demo Video

6 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()
```