Developing Pygame for Diamonds using GenAI

Vrinda Singhal
Mihika Khemka
Khushi Bansal

April 2024

1 Introduction

The Diamond Bidding Game is a two or three-player trick-taking card game with elements of bidding and bluffing. Here is a breakdown of the rules, gameplay, and strategies:

1.1 Components

Deck: A standard deck of cards (52 cards) is used, but only cards from Spades, Hearts, and Clubs are included (removing Diamonds).

Players: Two or three players compete to win the most points.

Bidding Cards: Each player has a separate suit assigned at the beginning (e.g., Spades for player 1, Hearts for player 2). Throughout the game, players secretly bid using cards from their assigned suit. No shuffling among the cards of different suits, just a completely random suit is assigned to every player.

1.2 Gameplay

Dealing: Separate suits are given to each player, no shuffling among cards of different suits.

Bidding Rounds: There are 13 rounds, one for each Diamond card (Ace to 2, with Ace being the highest value). In each round, a Diamond card is revealed. This card determines the points awarded to the winner of the round. Diamond cards come for bid in the following order: Ace(14 points) being highest, King and so on till 2 being lowest. Players secretly bid using a single card from their assigned suit. While bidding player don't know each other cards but once they bid for the particular diamond card of that particular round and scores are given then they reveal their bid simultaneously before the next round. Bids are revealed simultaneously. The player who bids the highest card value in their assigned suit wins the round and collects the points associated with the revealed Diamond card. Used cards are tracked to prevent reuse in future bids. Once a player uses a specific card from his/her assigned suit then it can't reuse that card again. In case of a tie (both players bid the same card value), the points are shared between the players.

Winning: The player with the highest score at the end of all 13 rounds wins the game. Points are accumulated by winning the bidding rounds and collecting the corresponding values of diamond cards.

1.3 Strategies

Tracking Used Cards: Keeping track of the cards your opponent has already used in their bids can help you predict their remaining high-value cards and adjust your strategy.

Bluffing with Mid-Range Cards: In early rounds, especially when bidding for high-value diamonds, you can consider using a mid-range card as a bluff if you suspect your opponent might overbid with a higher card they would like to save for later.

Prioritising High-Value Diamonds: Since the point value of the diamond card itself determines the score, it's often strategic to prioritise winning bids for the higher-value diamonds (Ace, King, Queen) in the earlier rounds. This secures a larger point advantage early on.

Calculated Risks for Lower Diamonds: For lower-value diamonds, you might take calculated risks with your bids. If you suspect your opponent does not have many high-value cards left, you might use a lower card to secure the win.

Adapting to Your Opponent: Observing your opponent's bidding patterns and the cards they haven't used can help you refine your strategy throughout the game.

Additional Notes: The game relies on a combination of strategy and luck, especially when dealing with random card draws and opponent bids. The diamond bid game can be played with variations in terms of the number of players, the point values associated with diamonds, and the strategic elements involved.

2 Problem Statement

We have to develop a diamond game frontend using Pygame with the help of GenAI, with the following UI features:
1. Card you are bidding
2. A row of cards in the bottom displaying all the cards you currently have in hand for bidding
3. Scoreboard showing the score for each round and the total score after every round
4. Computer bid after my bid

3 Teaching GenAI the game

We began by discussing the rules and mechanics of the Diamonds card game, including bidding strategies, win conditions, and the overall gameplay. We then translated these rules into Python code, implementing functions for bidding, playing rounds, and determining the winner. Throughout the conversation, we iteratively refined the code based on feedback and suggestions, addressing issues such as variable naming, logic errors, and game flow. The final code accurately simulates the Diamonds card game, allowing players to bid on diamond cards, play rounds against a computer opponent, and determine the winner based on the total face value of diamond cards won.

Teaching GenAI the Diamonds card game involves providing a comprehensive understanding of the game's rules, mechanics, and strategies. GenAI would need to learn how bidding works, the hierarchy of card values, and the win conditions. By breaking down the game into its constituent parts and explaining each aspect in detail, GenAI can grasp the intricacies of gameplay.

4 Iterating upon the code for UI

```python
import pygame
import random
import os

# Initialize Pygame
pygame.init()

# Constants
WIDTH, HEIGHT = 800, 600
CARD_WIDTH, CARD_HEIGHT = 60, 90
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
RED = (255, 0, 0)

```

```python
# Load Card Images
card_images = {}
suits = ['hearts', 'spades', 'clubs', 'diamonds']
for suit in suits:
    card_images[suit] = {}
    for value in range(2, 15):
        filename = f'{value}_of_{suit}.png'
        card_images[suit][value] = pygame.image.load(os.path.join(suit, filename))

# Create Player Class
class Player:
    def __init__(self, name, assigned_suit):
        self.name = name
        self.assigned_suit = assigned_suit
        self.hand = []
        self.used_cards = []

    def deal_hand(self, deck):
        self.hand = [card for card in deck if card[1] == self.assigned_suit]

    def bid(self, diamond_card):
        available_cards = [card for card in self.hand if card not in self.used_cards]
        return random.choice(available_cards)

# Create Window
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Diamond Bidding Game")

# Font
font = pygame.font.Font(None, 36)

# Main Function
def main():
    clock = pygame.time.Clock()
    running = True

    # Game Variables
    round_num = 1
    deck = [(value, suit) for value in range(2, 15) for suit in ['hearts', 'spades', 'clubs', 'diamonds']]
    random.shuffle(deck)
    player1 = Player("Player 1", "spades")
    player2 = Player("Computer", "hearts")
    player1.deal_hand(deck)
    player2.deal_hand(deck)

    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_SPACE:  # Press space to
                ↪  bid
                    player1_bid = player1.bid(deck[round_num -
                    ↪  1][0])
                    print(f"{player1.name} bids: {player1_bid}")
                    player2_bid = player2.bid(deck[round_num -
                    ↪  1][0])
                    print(f"{player2.name} bids: {player2_bid}")

        if not running:
            break  # Exit the loop if running is set to False

        screen.fill(WHITE)

        # Display Player's Hand
        x, y = 50, HEIGHT - CARD_HEIGHT - 20
        for card in player1.hand:
            suit = card[1]
            value = card[0]
            card_img = card_images[suit][value]
            screen.blit(card_img, (x, y))
            x += CARD_WIDTH + 10

        # Display Bidding Card
        bidding_card = deck[round_num - 1]
        suit = bidding_card[1]
        value = bidding_card[0]
        card_img = card_images[suit][value]
        screen.blit(card_img, (WIDTH // 2 - CARD_WIDTH // 2,
        ↪  HEIGHT // 2 - CARD_HEIGHT // 2))

        # Display Scoreboard
        text_round = font.render(f"Round: {round_num}", True,
        ↪  BLACK)
        screen.blit(text_round, (WIDTH - 150, 20))

        pygame.display.flip()
        clock.tick(30)  # Ensure smooth frame rate

        # Increase Round Number
        round_num += 1

        # Check if all rounds are completed
        if round_num > 13:
            running = False  # End the game loop

    pygame.quit()  # Quit Pygame

if __name__ == "__main__":
    main()
```

Initial Code:
The initial code provided the logic for a Diamond Bidding Game but lacked any graphical user interface (GUI). The game was entirely text-based, making it less engaging for players.

First Iteration (Initial GUI Setup):
Pygame, a popular Python library for game development, was introduced to create a graphical user interface (GUI) for the game. The code was modified to initialize Pygame and set up a display window using pygame.init() and pygame.display.set_mode(). Card images for Spades, Hearts, and Clubs were loaded using Pygame's image.load() function. Despite these changes, the GUI abruptly disappeared after a short while due to the lack of continuous updating within the game loop.

Second Iteration (Continuous Display):
The setup logic was moved inside the game loop to ensure continuous display of GUI elements (player's hand, bidding card, scoreboard) while the game is running. The display of the player's hand, bidding card, and scoreboard was incorporated into the game loop to update dynamically. Despite these improvements, the GUI still abruptly disappeared, and the game seemed to play automatically without player input.

Third Iteration (Proper Event Handling):
The game loop was adjusted to wait for user input for bidding, enhancing player interaction with the game. Event handling for user input (pressing the space bar to bid) was added to allow players to control the bidding process. Players were required to press the space bar to make bids, and bids were printed to the console. The GUI remained visible throughout the game, providing a more engaging and interactive experience for players.

Final Iteration (Additional UI Elements):
Additional UI elements were incorporated into the game to enhance the player experience further. Functions were added to display the player's hand, the bid-

ding card, and the scoreboard on the screen using Pygame's drawing functions. The computer's bidding behavior was simulated, and its bid was displayed on the screen after the player made a bid. These UI enhancements provided visual feedback to the player, making the game more intuitive and enjoyable.

Overall, through iterative improvements and the incorporation of UI elements using Pygame, the Diamond Bidding Game evolved from a purely text-based experience to a visually engaging and interactive game with enhanced player control and feedback.

5 Conclusion

The journey of iterating upon the code for the Diamond Bidding Game and incorporating a graphical user interface (GUI) using Pygame has been a valuable learning experience. Here's a final conclusion summarizing the key takeaways:

Enhanced User Experience: By adding a graphical user interface, we transformed the game from a text-based experience to a visually engaging one. Players can now interact with the game using mouse clicks or keyboard inputs, which significantly enhances their experience and immersion in the game.

Iterative Development Process: The process of iterating upon the code allowed us to gradually improve the game's UI and functionality. We started with a basic implementation and progressively added more features and refinements based on user feedback and requirements.

Importance of Proper Event Handling: Proper event handling is crucial for ensuring player interaction with the game. By implementing event handlers for user input, such as bidding, we enabled players to control the game flow and make decisions, resulting in a more dynamic and engaging gameplay experience.

UI Design Considerations: Incorporating UI elements, such as displaying the player's hand, bidding card, and scoreboard, required careful consideration of design principles and user interface guidelines. These elements were implemented in a way that is intuitive and easy for players to understand and interact with.

Continuous Improvement: The development process highlighted the importance of continuous improvement and iteration. Even after achieving a functional GUI, there are always opportunities to further enhance the game's UI, add new features, or optimize existing functionality to provide a better overall experience for players.