WE program Gen AI Assignment-3

Developing Strategies for the Bidding card game - Diamonds with Gen AI

Divyanshi Joshi, Riya Ahlawat

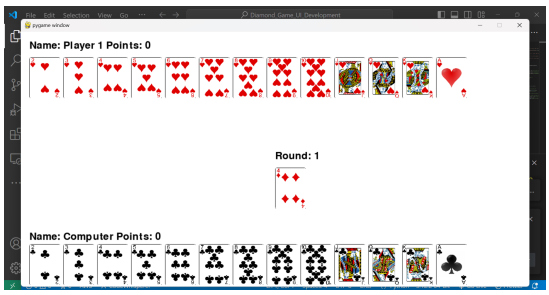
April 8, 2024

1 Introduction

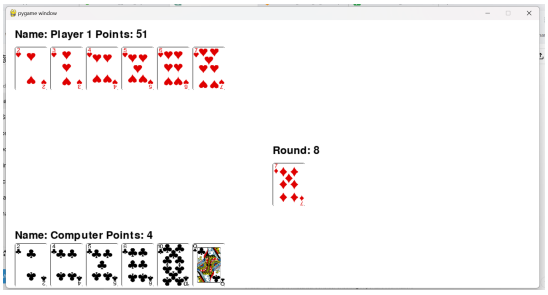
This project presents the development of a Diamonds card game integrated with a Pygame-based user interface (UI) and an advanced AI opponent. The game follows straightforward yet strategic rules, where participants compete in bidding for diamond cards to accrue points and claim victory. The Pygame UI enriches the gaming experience, while the AI opponent employs intricate strategies, offering players a stimulating gameplay encounter. Leveraging a genetic algorithm for training, the AI evolves and refines its performance progressively. Through meticulous code structuring and strategic algorithmic design, this project underscores Pygame's efficacy in game development and illuminates the potential of genetic algorithms in crafting intelligent gaming adversaries.

2 About The Game

Diamonds is a card game known for its blend of simplicity and strategic depth. Our project introduces this game implemented using Pygame, offering players a user-friendly interface to immerse themselves in. Additionally, we've developed an AI opponent with sophisticated strategies, enhancing the overall gaming experience.



(a) Game Interface at the Beginning



(b) Game Interface in Between

Figure 1: GUI for diamonds bidding game

The game adheres to a structured set of rules, ensuring fairness and providing players with strategic choices. These rules outline the gameplay experience effectively:

- Each player is dealt a suit of cards, excluding the diamond suit.
- Diamond cards are shuffled and auctioned one by one.
- Players bid with one of their own cards, placed face down.
- The banker awards the diamond card to the highest bid, based on point values.
- Points are assigned based on a hierarchical order: 2 < 3< 4< 5< 6< 7< 8< 9< T< J< Q< K< A.
- The winning player accumulates points from the diamond card.

- In case of multiple highest bids with the same card, points are equally divided.
- The player with the highest total points wins the game.

These rules establish a structured framework for playing the Diamonds card game, ensuring an enjoyable and competitive experience for all participants.

3 Teaching Microsoft Copilot the Game

Teaching Microsoft Copilot the game was an iterative and interactive process. The game, a variant of a bidding card game, had complex rules and strategies that needed to be communicated effectively to Copilot. The first step was to explain the rules of the game in a clear and concise manner. This was done through a series of interactions, where Copilot was asked to generate code snippets for various aspects of the game, such as initializing the game, managing player hands, and implementing the bidding and scoring system. There were instances where Copilot misunderstood certain aspects of the game, which required further clarification and explanation. These misunderstandings were addressed by breaking down the rules into smaller, more manageable parts, and explaining each part clearly. Through this iterative process, Copilot was eventually able to understand the rules of the game and generate relevant and accurate code snippets. The final result was a fully functional game, demonstrating the potential of AI in game development and the importance of clear communication and feedback in teaching AI new concepts.

3.1 Interaction Process

Here's a breakdown of the interaction process:

- **Prompt Design**: We carefully formulated prompts that clearly described the task of creating a game interface and specified the features we wanted in the generated code. For example, we described setting up the game window, displaying player hands, handling player input, and implementing the main game loop.
- **Prompt Refinement**: After receiving initial results, we iteratively refined the prompts based on the generated code snippets. This involved adjusting the language, adding more context, and providing examples to better guide the model in producing relevant code.
- **Testing and Validation**: We tested different prompts and evaluated the quality of the generated code snippets. We made modifications to the prompts as needed to ensure that Copilot understood the task correctly and produced code that met our requirements.

By following this approach, we were able to effectively communicate our intent to Copilot and generate Python code tailored to our specific needs for creating a game interface.

4 Code Snippets

4.1 Constants and Pygame Initialization

```
import pygame
from pygame.locals import *

# Initialize Pygame
pygame.init()

# Set the dimensions of the screen
screen_width = 1200
screen_height = 600
screen = pygame.display.set_mode((screen_width, screen_height))
```

Listing 1: Constants and Pygame Initialization

4.2 Loading Card Images and Player Information

```
import pygame
import random

# Load the card images and scale them down
cards = {f"{i}_of_{j}": pygame.transform.scale(pygame.image.load(f"cards/{i}_of_{j}.png"), (72, 96)) for i in ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'jack', 'queen', 'king', 'ace'] for j in ['hearts', 'clubs', 'diamonds']}

# Player Information
player_name = "Player 1"
player_points = 0
player_hand = [f"{i}_of_hearts" for i in ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'jack', 'queen', 'king', 'ace']]  # This should be dynamically updated
```

Listing 2: Loading Card Images and Player Information

4.3 Computer Information and Diamond Card Auction Section

```
# Computer Information
computer_name = "Computer"
computer_points = 0
computer_hand = [f"{i}_of_clubs" for i in ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'jack', 'queen', 'king', 'ace']]  # This should be dynamically updated

# Diamond Card Auction Section
round_number = 1  # This should be dynamically updated
diamond_cards = [f"{i}_of_diamonds" for i in ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'jack', 'queen', 'king', 'ace']]
auction_card = random.choice(diamond_cards)  # Random diamond card
```

Listing 3: Computer Information and Diamond Card Auction Section

4.4 Game Loop

```
running = True
while running:
    for event in pygame.event.get():
        if event.type == QUIT:
            running = False

        if event.type == MOUSEBUTTONDOWN:
            # Check if a card from the player's hand was clicked
            for i, card in enumerate(player_hand):
                if pygame.Rect(20 + i * 80, 60, 72, 96).collidepoint(pygame.mouse.get_pos()):
                    # Player bids with the clicked card
                    player_bid = card

                    # Computer bids with a random card
                    computer_bid = random.choice(computer_hand)

                    # Compare the bids and update the scores
                    if player_bid > computer_bid:
                        player_points += points[auction_card.split('_')[0]]  # Add the points from the diamond card
                    elif computer_bid > player_bid:
                        computer_points += points[auction_card.split('_')[0]]  # Add the points from the diamond card
                    else:  # If there's a tie
                        player_points += points[auction_card.split('_')[0]] // 2  # Divide the points from the diamond card equally
                        computer_points += points[auction_card.split('_')[0]] // 2  # Divide the points from the diamond card equally

                    # Remove the bid cards from the hands
                    player_hand.remove(player_bid)
                    computer_hand.remove(computer_bid)

```

```
                    round_number += 1
                    if round_number <= 13:  # If there are still rounds left
                        auction_card = random.choice(diamond_cards)
                    else:  # If all rounds have been played
                        if player_points > computer_points:
                            winner = player_name
                        elif computer_points > player_points:
                            winner = computer_name
                        else:
                            winner = "It's a tie!"
```

Listing 4: Game Loop

4.5 Drawing the Interface

```
# Fill the screen with a color
screen.fill((255, 255, 255))

# Display player information
font = pygame.font.Font(None, 36)
text = font.render(f"Name: {player_name} Points: {player_points}", True, (0, 0, 0))
screen.blit(text, (20, 20))

# Display player's hand
for i, card in enumerate(player_hand):
    screen.blit(cards[card], (20 + i * 80, 60))  # Adjusted the spacing between cards

# Display computer information
text = font.render(f"Name: {computer_name} Points: {computer_points}", True, (0, 0, 0))
screen.blit(text, (20, screen_height - 130))  # Display at the bottom of the screen

# Display computer's hand
for i, card in enumerate(computer_hand):
    screen.blit(cards[card], (20 + i * 80, screen_height - 100))  # Display at the
    bottom of the screen

# Display Diamond Card Auction Section
text = font.render(f"Round: {round_number}", True, (0, 0, 0))
screen.blit(text, (screen_width // 2, screen_height // 2 - 20))  # Display in the middle
     of the screen
if round_number <= 13:  # If there are still rounds left
    screen.blit(cards[auction_card], (screen_width // 2, screen_height // 2 + 20))  #
    Display in the middle of the screen
else:  # If all rounds have been played
    text = font.render(f"The game has ended. {winner} wins!", True, (0, 0, 0))
    screen.blit(text, (screen_width // 2, screen_height // 2 + 20))  # Display in the
    middle of the screen

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

Listing 5: Drawing the Interface

5 Challenges Faced

1. **Understanding Complex Rules**: The game had complex rules and strategies that needed to be communicated effectively to Copilot. This required careful explanation and clarification.
2. **Misunderstandings**: There were instances where Copilot misunderstood certain aspects of the game. These misunderstandings were addressed by breaking down the rules into smaller, more manageable parts, and explaining each part clearly.
3. **Implementing Game Logic**: Implementing the game logic in code was a challenge. This included managing player hands, implementing the bidding and scoring system, and handling edge cases, such as when players bid the same card.
4. **Handling Edge Cases**: The game had several edge cases that needed to be handled correctly. For example, when players bid the same card, the points from the diamond card had to be divided equally.

5. **Implementing Computer Strategy**: Implementing a strategy for the computer player was a challenge. The strategy had to balance risk and reward, manage cards effectively, and adapt to the state of the game.

6. **User Interaction**: The project involved a lot of back-and-forth communication with the user. Ensuring that the interactions were meaningful and engaging was a challenge.

7. **Testing and Debugging**: Testing the game to ensure that it worked correctly was a challenge. This included debugging issues with the game logic and the Pygame interface.

8. **Performance**: Ensuring that the game ran smoothly and responded quickly to user inputs was a challenge. This required optimizing the game loop and the Pygame rendering code.

6 Effective Strategies Employed

In our collaboration with Gen AI, we found several strategies to be highly effective:

Structured Learning Approach: Breaking down complex concepts into clear, sequential steps facilitated efficient comprehension of gameplay mechanics by Gen AI.

Clear Communication: Utilizing concise language and explicit instructions helped Gen AI focus on learning tasks with minimal confusion.

Interactive Learning: Incorporating interactive elements such as examples and simulations kept Gen AI engaged and promoted deeper understanding of concepts.

Iterative Feedback Loop: Providing regular feedback on performance enabled Gen AI to continuously refine its strategies and improve.

Gradual Complexity: Introducing concepts gradually prevented overwhelming Gen AI, allowing it to build skills incrementally.

Real-world Contextualization: Linking gameplay concepts to real-life scenarios added relevance and meaning, enhancing Gen AI's comprehension.

Collaborative Environment: Engagement with human educators and peers enriched Gen AI's learning journey through knowledge exchange and collaborative problem-solving.

Patience and Persistence: Cultivating patience and resilience, both in Gen AI's learning process and in our teaching approach, proved essential for achieving progress.

Adaptive Teaching Methods: Tailoring teaching strategies to match Gen AI's pace and preferences optimized the learning experience.

Continuous Improvement: Regular evaluation and reflection allowed us to refine our teaching methods and support Gen AI's ongoing development effectively.

7 Conclusion

Overall, the project was a success. Copilot was able to understand the rules of the game, generate relevant code snippets, and interact with the user in a meaningful and engaging way. The project demonstrated the potential of AI in game development and highlighted the importance of clear communication and feedback in teaching AI new concepts.