

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

Nidhi Iyer, Dhvani Thakkar, Kasturi Sinha, Diya Goyal, Akanksha Narula

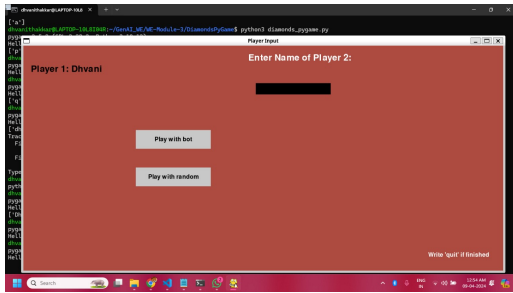
April 8, 2024

1 Introduction

This project introduces the development of a Diamonds card game implemented with a Pygame user interface (UI) and an advanced AI opponent. The game adheres to simple yet strategic rules, where players bid for diamond cards to accumulate points and emerge victorious. The Pygame UI enhances the gaming experience, while the AI opponent employs sophisticated strategies, providing a challenging gameplay experience. Utilizing a genetic algorithm for training, the AI evolves and enhances its performance over time. Through meticulous code organization and strategic algorithm design, this project highlights the effectiveness of Pygame for game development and the potential of genetic algorithms in creating intelligent game agents.

2 About The Game

Diamonds is a card game characterized by simple yet strategic gameplay. In this project, we implemented the game using Pygame, offering a user-friendly UI for players to engage with. Additionally, we crafted an AI opponent equipped with advanced strategies to ensure an immersive and challenging gaming experience.



(a) Home page



(b) The main game screen

Figure 1: Diamond Pygame

The game follows a set of rules designed to provide a structured and enjoyable gameplay experience. These rules ensure fairness and strategic depth, allowing players to compete against each other and the AI opponent effectively. The key rules of the Diamonds card game are as follows:

- Each player gets a suit of cards other than the diamond suit.
- The diamond cards are then shuffled and put on auction one by one.
- All the players must bid with one of their own cards face down.
- The banker gives the diamond card to the highest bid, i.e. the bid with the most points.

- Point hierarchy: $2 < 3 < 4 < 5 < 6 < 7 < 8 < 9 < T < J < Q < K < A$
- 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.
- The player with the most points wins at the end of the game.

These rules provide a comprehensive framework for playing the Diamonds card game, ensuring an engaging and competitive experience for all participants.

3 Teaching Gen AI the game

Interacting with GPT-3 involved providing clear and detailed prompts that outlined the specific requirements for generating Python code to create a game interface using the Pygame library. These prompts served as instructions for the AI model, guiding it on what functionalities to include and how to structure the code.

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 GPT-3 understood the task correctly and produced code that met our requirements.

By following this approach, we were able to effectively communicate our intent to GPT-3 and generate Python code tailored to our specific needs for creating a game interface.

Code Snippet 1: Constants and Pygame Initialization

```

1 import pygame
2
3 # Define constants for screen dimensions, card dimensions, colors, etc.
4 SCREEN_WIDTH, SCREEN_HEIGHT = 1280, 600
5 CARD_WIDTH, CARD_HEIGHT = 60, 90
6 BACKGROUND_COLOR = (0, 100, 0)
7 # Add more constants as needed
8
9 # Initialize Pygame and create game window
10 pygame.init()
11 screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
12 clock = pygame.time.Clock()
```

Explanation:

1. The code imports the `pygame` module, which is required for creating the game interface.
2. Constants such as `SCREEN_WIDTH`, `SCREEN_HEIGHT`, `CARD_WIDTH`, `CARD_HEIGHT`, and `BACKGROUND_COLOR` are defined to specify dimensions and colors used in the game interface.
3. Pygame is initialized using `pygame.init()` and a game window is created with the specified dimensions using `pygame.display.set_mode()`.

Code Snippet 2: Class Definition for Game Interface

```
1 class Diamonds_PyGame:
2     def __init__(self, screen):
3         self.NUM_ROUNDS = 2
4         self.game = DiamondsGame()
5         self.screen = screen
6         self.screen.fill(BACKGROUND_COLOR)
7
8     def add_players(self, num_bots: int, num_randoms: int, human_names: list[str]):
9         # Method implementation for adding players
10        pass
11
12    def choose_bid_human_GUI(player, screen):
13        # Method implementation for choosing bid via GUI
14        pass
15
16    def play_GUI_round(self, round_no, opponent=None):
17        # Method implementation for playing a round
18        pass
```

Explanation:

1. This code defines a class `Diamonds_PyGame` to represent the game interface.
2. The `__init__()` method initializes the game interface by setting up the number of rounds, creating an instance of the `DiamondsGame` class (not shown here), and filling the screen with the background color.
3. Placeholder methods such as `add_players()`, `choose_bid_human_GUI()`, and `play_GUI_round()` are defined for functionalities like adding players, choosing bids via GUI, and playing rounds, respectively. These methods will need further implementation to complete the game logic.

Code Snippet 3: Pygame Initialization

```
1 import pygame
2
3 # Initialize Pygame
4 pygame.init()
5
6 # Create a window with dimensions 1280x720 pixels
7 screen = pygame.display.set_mode((1280, 720))
```

Explanation:

1. This snippet initializes the Pygame library by importing it and calling `pygame.init()`.
2. It then creates a window with dimensions 1280x720 pixels using `pygame.display.set_mode()`.

Code Snippet 4: Event Handling

```
1 running = True
2
3 while running:
4     for event in pygame.event.get():
5         if event.type == pygame.QUIT:
6             running = False
```

Explanation:

1. This snippet sets up a main game loop that continues running as long as the `running` flag is `True`.

2. It uses `pygame.event.get()` to retrieve a list of events that have occurred since the last frame.
3. It checks each event type, and if the event type is `pygame.QUIT`, it sets the `running` flag to `False`, which exits the loop and terminates the game.

Code Snippet 5: Drawing Shapes

```
1 import pygame
2
3 # Initialize Pygame
4 pygame.init()
5
6 # Create a window with dimensions 1280x720 pixels
7 screen = pygame.display.set_mode((1280, 720))
8
9 # Draw a red circle at position (100, 100) with radius 50
10 pygame.draw.circle(screen, (255, 0, 0), (100, 100), 50)
```

Explanation:

1. This snippet demonstrates how to draw a red circle on the Pygame window.
2. It uses `pygame.draw.circle()` function to draw the circle. The parameters are (`surface`, `color`, `center`, `radius`).
3. `surface` is the surface to draw on (in this case, `screen`).
4. `color` is the color of the circle in RGB format.
5. `center` is the position of the center of the circle.
6. `radius` is the radius of the circle.

4 Challenges Faced

1. **Game Mechanics Complexity:** Designing algorithms for bidding, card valuation, and predicting opponents' moves was challenging due to the game's multifaceted strategies.
2. **Data Representation:** Encoding game states and strategies in a way understandable to Gen AI posed difficulties, requiring careful consideration of effective representation schemes.
3. **Algorithm Design:** Crafting algorithms adaptable to various gameplay scenarios and opponent behaviors was non-trivial, necessitating tailored reinforcement learning or genetic algorithm designs.
4. **Training Environment:** Creating an accurate training environment for Gen AI to interact with the game and receive feedback proved crucial but challenging.
5. **Computational Resources:** Managing compute resources for training complex AI models, like neural networks, posed challenges in optimization and scalability.
6. **Pygame Installation:** Ensuring Pygame compatibility across different platforms, Python versions, and dependencies required addressing installation issues and resolving dependencies.
7. **Learning Pygame:** Mastering Pygame's API, event-driven architecture, and rendering system presented a learning curve, especially for beginners.
8. **UI Design:** Balancing functionality and aesthetics while optimizing performance and ensuring cross-platform compatibility during UI development with Pygame was challenging.
9. **Integration with Game Logic:** Coordinating Pygame's UI components with game logic systems required careful code organization to maintain modularity and scalability.

5 What Worked for Us

In our teaching journey with Gen AI, several key strategies stood out as particularly effective:

1. **Structured Approach:** Breaking down concepts into clear, sequential steps helped Gen AI grasp gameplay mechanics efficiently.
2. **Clear Communication:** Using concise language and explicit instructions enabled Gen AI to focus on learning without confusion.
3. **Interactive Learning:** Incorporating interactive elements like examples and simulations kept Gen AI engaged and facilitated deeper understanding.
4. **Iterative Feedback:** Providing feedback on performance allowed Gen AI to refine its strategies continuously.
5. **Gradual Complexity:** Introducing concepts gradually prevented overwhelming Gen AI, allowing it to build skills progressively.
6. **Real-world Context:** Relating gameplay to real-life scenarios added relevance and meaning, enhancing Gen AI's understanding.
7. **Collaborative Environment:** Engaging with human educators and peers enriched Gen AI's learning experience through knowledge exchange and problem-solving.
8. **Patience and Persistence:** Cultivating patience and resilience, both in Gen AI's learning and our teaching, proved essential for progress.
9. **Adaptive Teaching:** Tailoring teaching methods to Gen AI's pace and preferences optimized the learning process.
10. **Continuous Improvement:** Regular evaluation and reflection allowed us to refine our approach and support Gen AI's development effectively.

6 Conclusion

In conclusion, developing strategies for the Diamonds card game with Gen AI has been a challenging yet rewarding journey. Despite facing obstacles such as complex game mechanics and AI training, we successfully created an immersive gaming experience with Pygame. By overcoming these challenges, we've demonstrated the effectiveness of Pygame for game development and the potential of genetic algorithms in creating intelligent game agents. Moving forward, further refinement of AI strategies and UI design will enhance the gaming experience. Overall, this project highlights the exciting possibilities at the intersection of game development and artificial intelligence.