Women Engineer's program Gen AI Assignment-6

Polishing a Diamond

PyGame GUI & Code Optimization with the help of GenAI

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

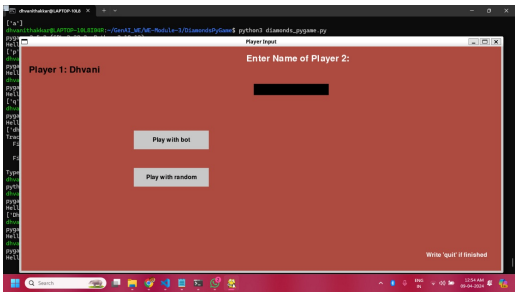
April 8, 2024

1 Introduction

This project introduces the development of a Diamonds card game implemented with a PyGame user interface and an advanced bot 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 bot algorithmic opponent employs sophisticated strategies, providing a challenging game-play experience. Utilizing a strategic algorithm developed with the help of GenAI, the bot plays extremely well against humans and random choice players. Through meticulous code organization and strategic algorithm design, this project highlights the effectiveness of PyGame for game development and the importance of readable, clean, modular and changeable code in web/app/game development.

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 algorithmic 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 strategy ideas and sub-parts, 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 Gemini/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

```
import PyGame

# Define constants for screen dimensions, card dimensions, colors, etc.
SCREEN_WIDTH, SCREEN_HEIGHT = 1280, 600
CARD_WIDTH, CARD_HEIGHT = 60, 90
BACKGROUND_COLOR = (0, 100, 0)
# Add more constants as needed

# Initialize PyGame and create game window
PyGame.init()
screen = PyGame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
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

```
class Diamonds_PyGame:
    def __init__(self, screen):
        self.NUM_ROUNDS = 13
        self.game = DiamondsGame()
        self.screen = screen
        self.screen.fill(BACKGROUND_COLOR)

    def add_players(self, num_bots: int, num_randoms: int, human_names: list[str])
        # Method implementation for adding players
        pass

    def choose_bid_human_GUI(player, screen):
        # Method implementation for human to choose bid via GUI
        pass

    def play_GUI_round(self, round_no, opponent=None):
        # Method implementation for playing a round
        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: Diamonds Game Functionality

```
class DiamondsGame:
  def __init__:
    pass

  def add_human_player:
    pass

  def add_bot:
    pass

  def add_random:
    pass

  def setup_game:
    pass

  def play_round:
    pass
```

Explanation:

1. A DiamondsGame class object is used as an attribute of PyGame class.
2. It all the add player functions are used once player configuration choice is collected from the GUI. Function play_round is completely modified to add GUI as play_GUI_round function in PyGame class.

Code Snippet 4: Event Handling

```
running = True

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

Explanation:

1. This snippet sets up a main game loop that continues running while the `running` flag is `True`.
2. It uses `PyGame.event.get()` to retrieve a list of events 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: Play GUI Round

```
def play_GUI_round(self, round_no, opponent = None):
    """Display the game state on the screen"""
    self.screen.fill(BACKGROUND_COLOR)
    print_round_title(self.screen, round_no)

    diamond = self.game.diamond_pile.pop(0)
    self.game.revealed_diamonds.append(diamond.value)

    if opponent:
        opponent_hand = opponent.get_hand_values()

    bids = []
    highest_bid = 0
    winners = []

    for player in self.game.players:
        if player.isBot and opponent_hand:
            diamonds_seen = self.game.revealed_diamonds
            bid = player.choose_bid(diamond, opponent_hand, diamonds_seen)
        elif player.isRandom:
            bid = player.choose_bid()
        else:
            # Display cards in the players' hands
            screen.fill(BACKGROUND_COLOR)
            print_round_title(self.screen, round_no)

            diamond.display_card(screen, CARD_WIDTH, CARD_HEIGHT)
            display_player_hand(player, CARD_WIDTH, CARD_HEIGHT, self.screen)

            bid = self.choose_bid_human_GUI(player, self.screen)

        bids.append(bid)

        if bid.value > highest_bid:
            winners = [player]
            highest_bid = bid.value
        elif bid.value == highest_bid:
            winners.append(player)

    display_bids_winners(self.screen, bids, self.game, round_no, diamond)
```

Explanation:

1. Iterates through each player:
2. For AI players with opponent, uses opponent's hand information for bidding with choose_bid.
3. For random players, uses choose_bid for a random bid.
4. For human players:
   - Displays cards.
   - Calls choose_bid_human_GUI to capture their bid through the GUI.
5. Tracks the highest bid and winner(s).
6. Displays all bids, players, winners, highest bid, round number, and revealed diamond value using display_bids_winners.

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 algorithm to various game-play scenarios and opponent behaviors was non-trivial, necessitating tailored conditional behaviours for the diamonds left, remaining card strength of opponent vs bot and relative strength of revealed diamond card. This eventually garnered a 78.6% win rate of the bot against a random choice player.
4. **Modularity using a new module:** Knowing when to initialise pygame, make a new screen, or clear the screen in different classes and functions used proved challenging.
5. **PyGame Installation:** Ensuring PyGame compatibility across different platforms, Python versions, and dependencies required addressing installation issues and resolving dependencies.
6. **Learning PyGame:** Mastering PyGame's API, event-driven architecture, and rendering system presented a learning curve, especially for beginners.
7. **UI Design:** Balancing functionality and aesthetics while optimizing performance and ensuring cross-platform compatibility during UI development with PyGame was challenging.
8. **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 long drawn code, 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 effectiveness of a simple statistical and conditional algorithm. Moving forward, further refinement of the UI design will enhance the gaming experience, while more modularity in the GUI functions, with a clearer overall flow will further improve the code. Overall, this project highlights the incredible ability of learning an absolutely new framework for making a nontrivial project with the help of genAI and incorporating existing code & modularisation for clean and immersive GUI.