**REPORT ON THE "MATCH CARDS" GAME IMPLEMENTATION**

Name: Sindhu Buggana

CWID: 20020990

The provided code implements a card game called "Match Cards" using Object-Oriented Programming (OOP) concepts in Python. The game is played between two players, and the objective is to match as many pairs of cards as possible within a certain number of turns. The code includes several classes to represent the different components of the game, such as cards, decks, hands, and the game itself.

Classes

1. Card: This class represents a single card in the game. It has attributes for the suit and rank of the card, as well as methods for string representation and equality comparison.

2. Deck: This class represents a deck of cards. It initializes the deck with all the standard cards, including two Joker cards, and provides methods for shuffling and dealing cards.

3. Hand: This class represents a player's hand of cards. It inherits from the `Deck` class and provides methods for adding cards to the hand and displaying the hand.

4. CardGame: This is a base class for card games. It initializes the deck and shuffles it.

5. MatchCardsGame: This is the main class that implements the "Match Cards" game logic. It inherits from the `CardGame` class and includes methods for handling game turns, scoring, and special moves.

6. OldMaidHand: This is a subclass of `Hand` that provides a method for removing matched pairs of cards from the hand.

7. CardValueTracker: This class is responsible for tracking the count of each rank value in the deck.

Game Flow

The game flow is controlled by the `play\_game` method in the `MatchCardsGame` class. Here's a summary of the game flow:

1. The game starts by initializing the player hands and setting up the game parameters based on the chosen difficulty level.

2. Each turn, a card is dealt to the current player, and the game checks for special cases, such as drawing a Joker card or matching a card with the opponent's hand.

3. If a player draws a Joker card, they can choose to match any card from their opponent's hand.

4. If a player matches a card with their opponent's hand, the opponent's score is incremented, and the matched card is removed from the opponent's hand.

5. The game also checks for special card combinations or sequences that trigger unique actions or bonuses, such as earning an extra turn or swapping cards with the opponent.

6. The scoring system can be either normal (one point per matched card) or based on the rank value of the matched card, depending on the difficulty level.

7. After all turns are completed, the game displays the final scores and declares the winner.

Additional Features

The code includes the following additional features:

1. Difficulty Levels: The game allows the user to select different difficulty levels (easy, medium, hard), which affect the number of turns, the number of Joker cards, and the scoring system.

2. Joker Cards: The game includes two Joker cards, which act as wildcards and can match any other card in the opponent's hand.

3. Card Tricks or Special Moves: The game introduces special card combinations or sequences that trigger unique actions or bonuses, such as earning an extra turn or swapping cards with the opponent.

Overall, the code demonstrates a good understanding of OOP concepts and provides a comprehensive implementation of the "Match Cards" game with additional features to enhance the gameplay experience.

Conclusion

The provided code successfully implements the "Match Cards" game using Object-Oriented Programming (OOP) concepts in Python. The implementation covers all the basic requirements of the game, such as dealing cards, matching cards with the opponent's hand, keeping track of scores, and determining the winner.

Additionally, the code includes several enhancements and extra features to make the game more engaging and challenging. The introduction of difficulty levels allows players to adjust the game's complexity based on their preferences. The inclusion of Joker cards as wildcards adds an interesting twist to the gameplay, requiring players to strategize their moves carefully.

Furthermore, the implementation of special card combinations and sequences that trigger unique actions or bonuses, such as earning an extra turn or swapping cards with the opponent, adds an element of surprise and excitement to the game. These features help to maintain the players' interest and encourage them to think critically about their card selections.

Overall, the code demonstrates a solid understanding of OOP principles and effective implementation of game logic. The modular design and clear separation of concerns make the code easy to understand, maintain, and extend with additional features or modifications in the future.

While the code appears to be functional and well-structured, it is important to note that thorough testing and user feedback would be beneficial to identify and address any potential issues or areas for improvement. Additionally, incorporating a graphical user interface (GUI) could enhance the overall user experience and make the game more visually appealing.

In conclusion, the "Match Cards" game implementation presented in this code is a commendable effort, showcasing the developer's skills in Python programming, OOP concepts, and game design. With its engaging features and potential for further enhancements, this code can serve as a solid foundation for future iterations or similar card game projects.