**Report on UNO Game Development**

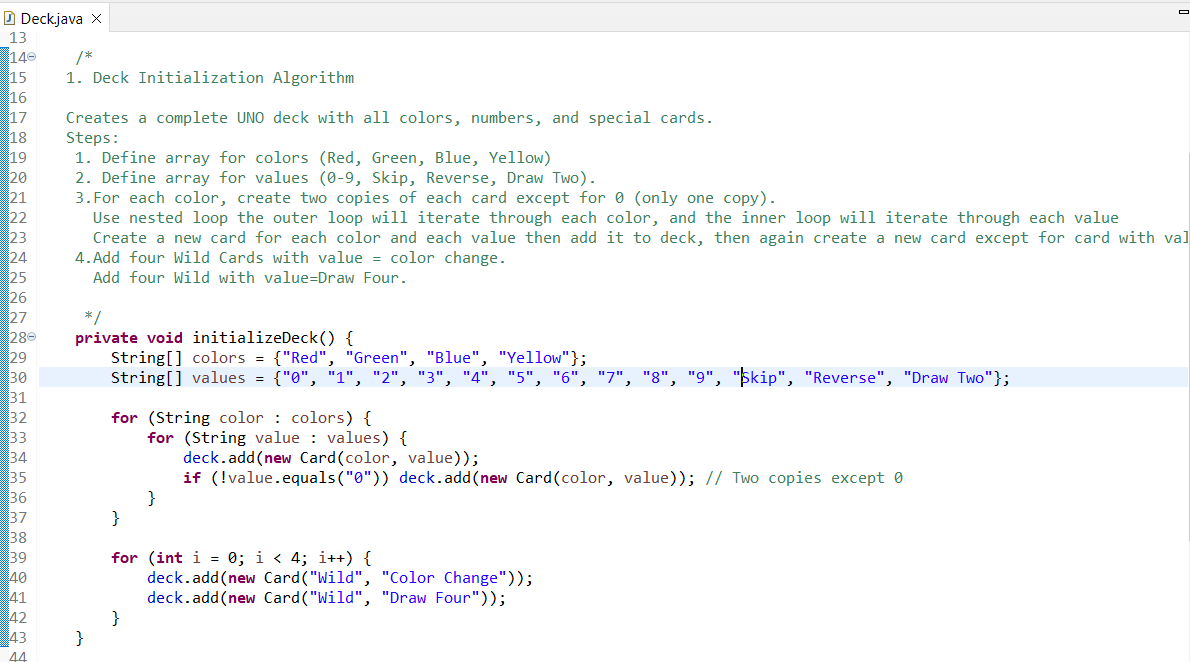
**1. Algorithms and Their Usage**

Card deck management (in Deck class), game flow, and handling special card effects (in Game class) are the main algorithms developed for the UNO game. The UNO card deck is shuffled by the Deck class algorithm, which guarantees card distribution randomness. This is accomplished by iterating through the card list and switching each card out for another at a randomly chosen index using a random shuffle technique. By ensuring that the deck is shuffled to replicate real-world randomness, this technique avoids predictable results.

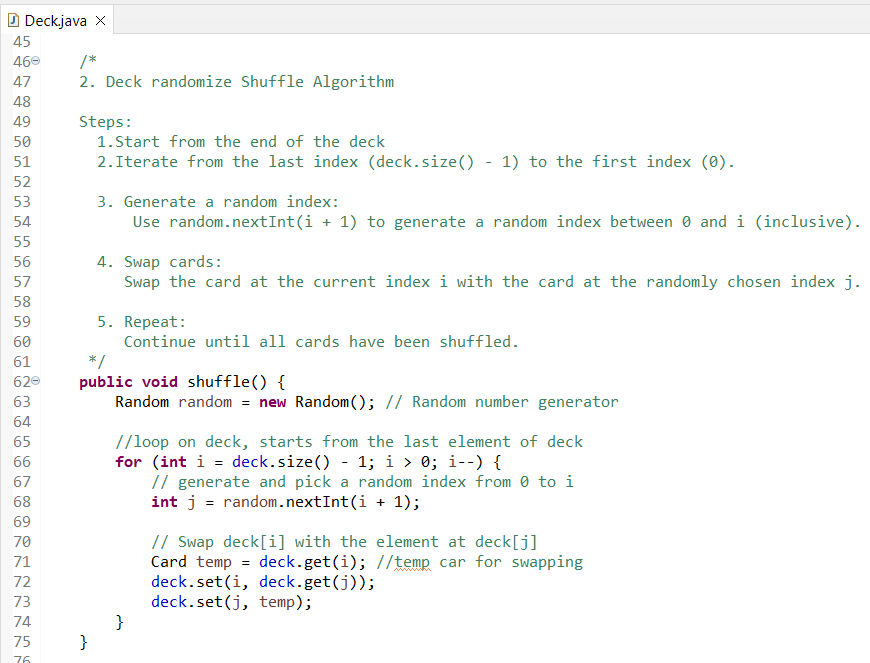
The game turn flow, which is managed by the Game class, is another important algorithm. This algorithm controls player turns, enforces game rules (such matching colors or numbers), and looks for special cards (like Skip, Reverse, and Draw Two). The algorithm determines if the move is legal by examining the card that is currently in play and comparing it to the player's card selections. The associated effect (such as skipping a player's turn) is carried out if a player plays a special card.

Finally, a card-handling algorithm is used in the drawing and returning of cards to the deck, guaranteeing fair gameplay and accurate card distribution. Players draw cards at random from the deck, and when a card is discarded or the game is over, it is either shuffled or put back into the deck for use later.

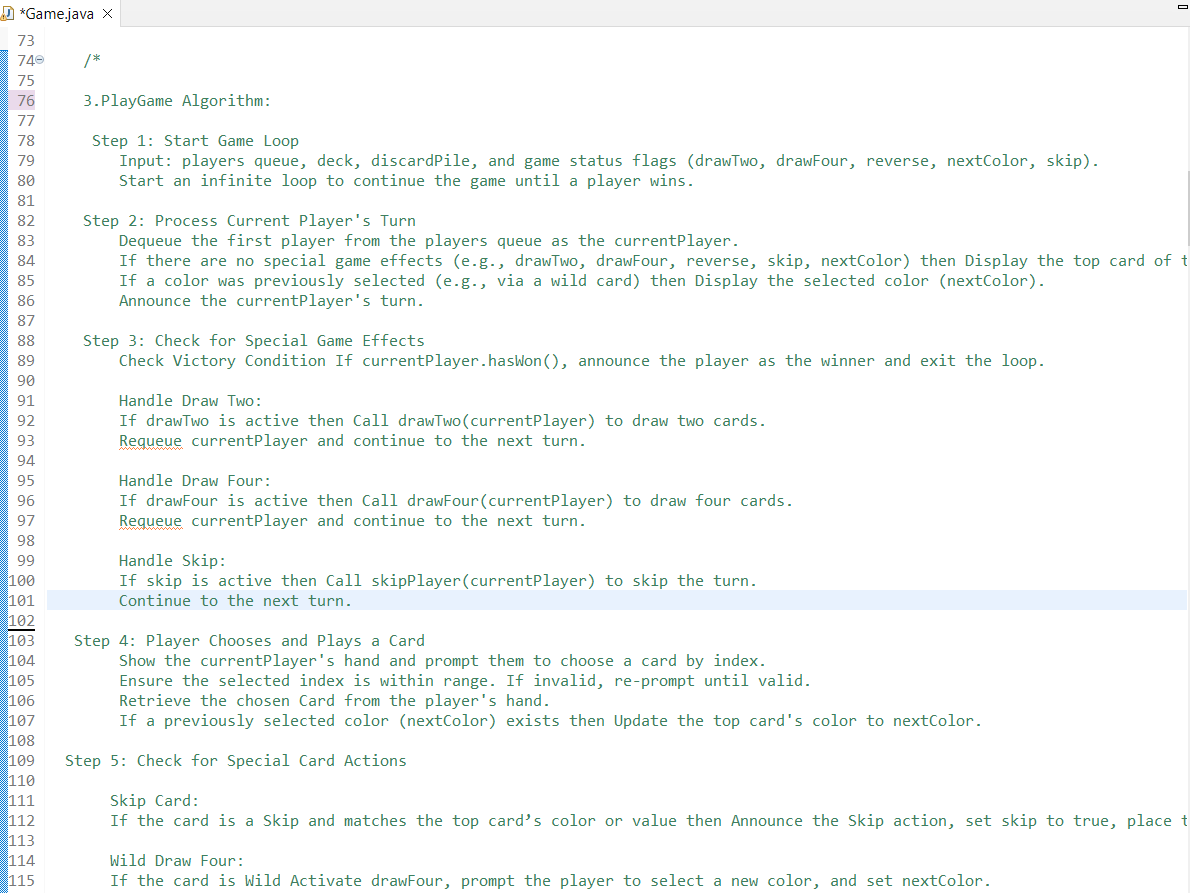
**Algorithm1 Screenshot:**

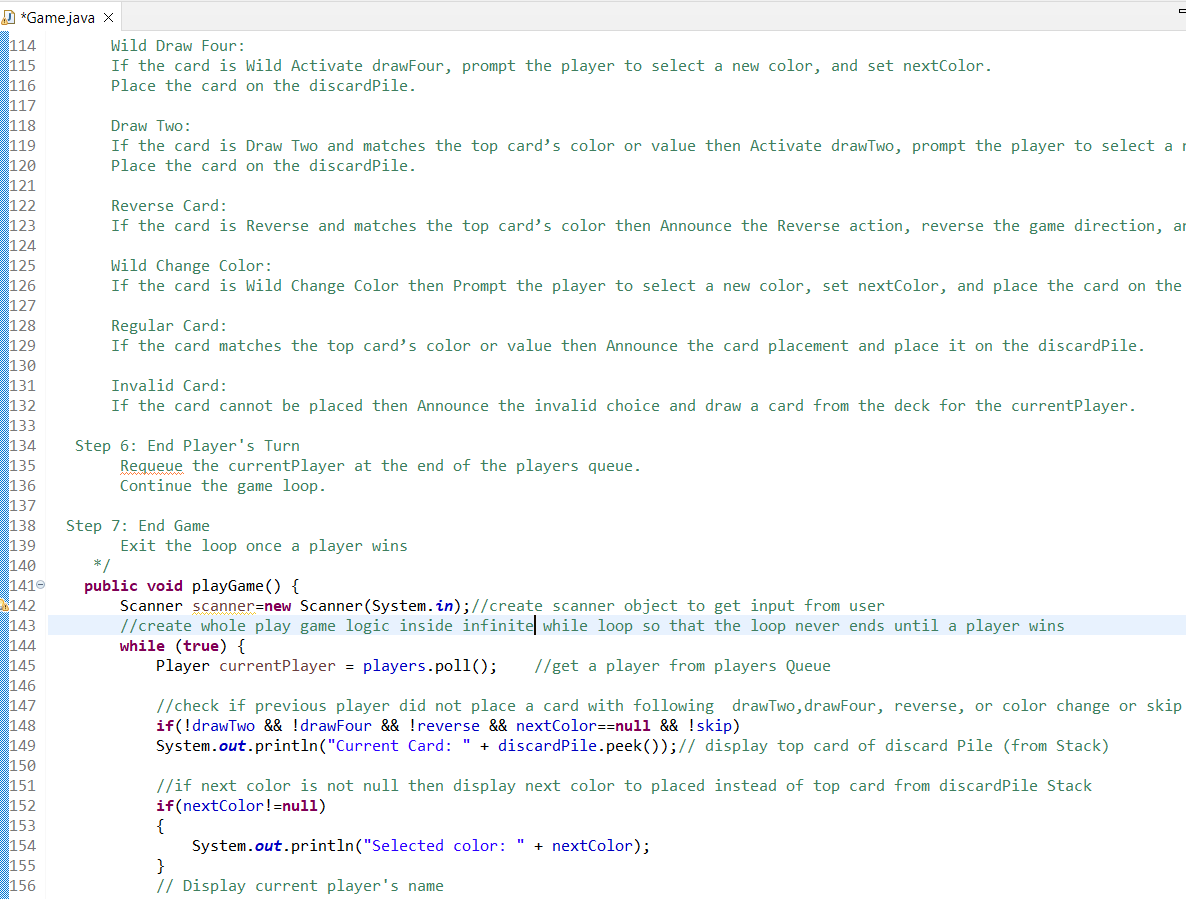


**Algorithm2 Screenshot:**



**Algorithm3 Screenshot:**





**2. Big O Time Complexity**

* **Shuffling the deck:** A card is switched out for a randomly selected card at each iteration of the shuffling algorithm, which is a variant of the Fisher-Yates shuffle. **O(n),** where n is the number of cards in the deck, is the time complexity for this operation. Since a fair shuffle requires that each card be touched once, this is ideal.
* **Drawing cards:** Since choosing a card and taking it out of the deck has a time complexity of O(1), drawing a card from the deck is basically a constant time operation.
* **Handling turns and special card effects:** A player's turn and special cards are processed by the game logic, which compares the current card with the player cards that are accessible and then performs the appropriate action (such as skipping a turn). The worst-case temporal complexity of this step is O(n), where n is the number of participants or actions that need to be checked.

As a result, the total time complexity of the game is highly efficient since it largely depends on the number of players (for managing turns and game flow) and cards (for shuffling).

**3. Data Structures Used**

Several data structures were employed throughout the UNO game to efficiently handle game operations:

* **List (or ArrayList):** The deck of cards and players’ hands are stored in arrays (Java). This choice allows for fast access and modification of cards, particularly when drawing, playing, or shuffling cards. I used ArrayList because it dynamically adjusts its size as cards are added or removed from a player's hand, which is necessary for an UNO game.
* **Queues:** A queue could be used for handling the turn order. A queue would be ideal for tracking players in a circular turn order. I used Queue to store player because the Queue ensures that players take their turns in the correct order (FIFO: First-In, First-Out).
* **Stacks:** Stacks are useful for handling the discard pile, where the top card is always accessible. This ensures that the last discarded card is always used to check for valid moves. I used Stack to store Discard Pile because A Stack inherently provides Last-In, First-Out (LIFO) functionality, where the most recently played card (push()) is the first card retrieved (peek() or pop()).

These data structures were chosen due to their efficiency in supporting the game’s requirements, such as quick access and removal (lists), maintaining turn order (queues), and managing the discard pile (stacks).

**4. Opportunity in the Development Process**

One key opportunity that arose during the development process was the introduction of special card effects (Skip, Reverse, Draw Two). Initially, the game only allowed for basic card matching based on color or number. However, I recognized the opportunity to enhance the gameplay experience by adding these special cards, which would introduce more strategy and excitement to the game. This addition required modifying the game flow to handle these unique cards’ effects. The ability to add this feature not only improved the game but also deepened my understanding of handling game rules and interactions.

**5. Encountering an Error and Resolving It**

Card matching on the player's turn was one of the major mistakes made throughout development. In particular, it was occasionally possible for players to play a card that did not match the color or number of the card being played, which was against the rules of the game. This mistake happened as a result of improperly implemented logic that caused the system to ignore specific situations when verifying the validity of the card. I fixed this by going over the card comparison logic again and adding more rigorous tests to make sure the card that was played was either a legal special card, the same color, or the same number. To address edge circumstances where the rules could be misunderstood, the Game class's validation function was improved.

**6. Changes for Future Versions**

In the next version of the game, I would focus on adding more advanced features, such as:

* **Multiplayer Support using network:** While the current version is designed for multi-player but on single pc, the next version could include networked multiplayer capabilities, allowing users to play against other people over the network. This would require socket programming logic and an updated architecture to manage player connections and game states.
* **User Interface Enhancements:** The current UI is basic and could be improved by adding GUI with visual effects for special cards, animations for card drawing, and a more immersive experience overall.
* **AI for Single-Player Mode:** Implementing AI opponents would allow users to play solo against intelligent computer players. This would require algorithms for decision-making and difficulty adjustment to keep the game challenging.