Final Project: Uno

Ever since my older cousin introduced me to a card game called "Yu-Gi-Oh," I have been a big fan of card games. Anything from trading card games, to casual games like goldfish, and more complicated games like poker. I knew I wanted to try and implement a card game for my project but I was stumped on what game to pick. Originally I thought of doing solitaire because it was a single-player game that seemed like it would be very simple. My mind changed when I started receiving a few messages in my extended family group chat, making plans for the coming Thanksgiving holiday the next week. Among other things we were talking about what games we should bring to play as a family and "Uno" was brought up. I began thinking of possible ways to implement the game, utilizing the topics I learned in my cs20 course. Many things clicked almost perfectly, and there were more features I thought of that would make the project more complex than Solitaire, so Uno became my game of choice.

The first issue I had was shuffling the deck, I had to utilize the "ctime" library and random RNG seeding to shuffle the deck. The deck itself utilized the Deque ADT, because I wanted to allow the user to draw from either the top or the bottom of the deck. Initially, I was having trouble matching the "gameDeck's" Deque structure to the "gamePile's" Array stack structure, I realized that I had not connected my Deque's pointers correctly. Another issue that took me a lot of time to solve was accidentally coding isEmpty() incorrectly for the Deque class. This prevented me from working on the deck and game logic for a long time, but after utilizing throwing and catching exceptions I was able to pinpoint and solve the problem. The array stack was chosen for the gamepile, because a stack was a perfect choice and I was lacking array-based ADTs. When playing Uno in real life, the cards that are used each turn are placed on a stack, which keeps getting larger until, either someone wins or the deck runs out.

There were a few issues I had to consider when I decided on the data structures for the project. First I had to figure out how the program would know which player's turn it was, simply moving through the list of players could be done by many structures but there were two specific things that needed extra consideration. The game would be going on for multiple rounds, meaning that once the last player in the order had their turn, the first player would get another turn, so I needed a way to go from the end to the beginning. Also, the reverse card existed within the game meaning the order could be flipped after any turn so a way to reverse the order was also needed. The data structure I figured would work best was a doubly linked list. The reason I needed the list to be doubly linked was to allow the reverse card to work when the order has reversed the node before the current node could be accessed to implement a reversal. And the Linked List allowed me to access any of the Players at any point, unlike some other structures that will only allow the top or bottom elements to be accessed.

Another issue that took me much longer than I expected was coding the main program. The logic of Uno seems so simple, but so obnoxious when put into code. I opted for a menu-based program where the user could play, see rules, or quit the program, and even within that, there were multiple menus. There were menus for which card you wanted to play and what move you were going to make during your turn. Because of the multiple menu choices, and no way to stop the user, they could incorrectly pick a choice such as playing a card when they had no possible cards to play. To solve this issue I used plenty of do-while statements to validate the inputs the user was putting before moving forward. One big example of this is on the menu displayed when it is the user's turn. The user can: Play a card, draw a card, sort their hand, or forfeit. If the user decides to Play a card, the program checks to make sure they have a valid card to play, and if they don't then I decided to force them to draw. Though, even when drawing I tried to give the player some agency, since playing against the "COM" players within my program seemed much less enjoyable than playing Uno in person.

The two completely new classes that I created during this project ended up being extremely useful. They were specifically built to solve the issues that the ADTs in class could not solve. The card class allowed me to consolidate the color and value attributes of the cards into one object and suppress some of the large amounts of code on main.cpp within its class functions. The player class allowed me to combine the card class and the LinkedList data structure into one object again, but I feel like I could have done more with the player class. It showed me firsthand how flexible code can be because this has been the largest coding project I have done so far. I feel happy with what I completed, but I feel like I could improve it, I am considering adding new rules to the game to try and match the feeling of playing Uno with friends or family in real life.