Instructions and Testing

When you run chess.py, you will start at the menu with 4 options. Option 1 starts a game of chess. Option 2 brings up a URL to the rules of chess Wikipedia if you are unfamiliar with chess. Option 3 brings up instructions on how to use the app. Option 4 exits the app. Simply enter 1,2,3 or 4 to the input prompt to make your selection. Once you are in a game of chess, indicate the move you would like to make by first inputting the space containing the piece you would like to move, and then input the space you would like to move it to. An example opening move would be to first input 'e2' and then input 'e4' to move the pawn from space E2 to E4. The user will receive feedback if a move is invalid or if an input doesn't make sense. To decipher which pieces the acronyms represent, please reference the "Piece Keys.txt" file. There are 3 moves this simulator is not capable of. The first is castling, the second is en passant, and the third is when a pawn reaches the other end it is not replaced with a queen. There is also no checker for check or checkmate and the king must be captured for the game to end. Besides this, all valid chess moves are possible! At any point in the game, you can type "quit" in the input to exit back to the menu.

For testing the project, you can play any chess games that does not include any of the moves mentioned above. An example of a game that follows that format is included in the "No Castle Game.txt" document. Follow that game for testing this program. You can also input whatever other chess moves you would like and input commands that make no sense to test the input checking. Upon a victory, the program will say who won and return to the main menu.

Work Completed / To be Finished Later

In this project, I completed a playable chess game with a home screen menu and user input checking. This project includes nine classes. The board class contains all the pieces and their locations and is printed to the screen for the user to view. The Piece class is the superclass to the Pawn, Rook, Knight, Bishop, Queen and King classes. Each of these classes define how the pieces are allowed to move on the Board class. Finally, the menu class handles the main menu that is created when running the program.

With more time and more lines of code available, I would incorporate the rules of chess not currently incorporated. This includes castling, en passant, and replacing a queen with a pawn when it reaches the other end. The other thing I would do is implement Unicode symbols instead of acronyms for the pieces. This was my original intent, but I had trouble implementing Unicode outside of Jupyter Notebooks. I would also include a function that looks for check and checkmate to eliminate illegally putting yourself in check and also so you don't have to capture the king to win the game.

Challenges

I had two major challenges with this project. The first was redesigning the __repr__ function for the board class. I originally designed my code in Jupyter Notebooks which included

Unicode representation of the pieces. I had to redesign the spacing of the board to incorporate the acronym representation. The other challenge I had was the length of the program. My first draft of the program had close to 1500 lines of code which is much more than the requirement of 750. I was able to increase the modularity of the code to reduce it down to 841 lines (when you take away whitespace and comments it meets the requirement).