**Script**

This is my testing video for Secret Agent Prep: Chess. To start with the testing process, I’ll first make an account after running the code. I’ll give the username “testing” and the password will be “testing101”. This should then store the username and password in the database so if the user runs the code again, it should allow them to login and access the Main Menu Screen with all of the options available. I will now demonstrate the login system using the details that we have just created.

And, as you can see, the login system has been successful, and I’ve re-directed the user to the Main Menu Screen. This process is called normal testing because this is a test to show that when appropriate data is entered, we get the expected output. However, I just want to demonstrate that if the user entered some details that aren’t in the database, it wouldn’t allow them to advance further. So, I will first typo the username to show that an error is returned if this happens before doing the same thing with the password. This process is called boundary testing. I will then enter something in both fields that isn’t remotely similar to the account details created by the user. This is called erroneous testing. I have also added one last bit of error detection. To ensure that the user doesn’t create an account with the same username, I’ve added an additional check to ensure that the username that is entered by the user doesn’t already exist in the database and I will demonstrate this now by attempting to create an account called “testing”.

Upon successfully creating a new account, the user will be directed to this screen, the Main Menu Screen. Here you can see that there are four buttons, which, when I was fetching my requirements from my end-client, was one of his requirements instead of the drop down menu style user interface, which is present on the currently existing systems like lichess.org and chess.com. One button is the back button, which upon clicking, will direct the user back to the previous screen. The other three buttons are related to the functionality of the games themselves.

The first button I will be testing is the standard chess variant. Upon clicking on the button, the window should close down and there will be some text in the terminal asking for certain inputs to play the game. The first input required is the background colour, followed by the square colours and then the piece set that the user wants. This style was inspired by the system that is already present on chess.com where the user is able to customise all features based on the available options on the site. I have added various error detection mechanisms for picking the colour schemes for the game so as to ensure that the input is valid and in the cases of certain colours, I have also not added them to the list (like black) for purely design purposes as you will not be able to see the black pieces that have been captured. I have also added this for the piece sets to ensure that a valid piece set has been selected (which I have chosen to number between 1 and 4 so that the user has a better experience and is not having to type out the piece set name). Upon entering a number, I have created an algorithm that does an O (1) search for the corresponding piece set from the number that has been inputted. This is the normal game of chess as we know it. There are a total of 6 different pieces: pawns (each side has 8), knights (each side has two), bishops (each side has two), rooks (each side has two), queens (each side has 1), kings (each side has one). I will now demonstrate that the pieces are all able to make their respective legal moves. The pawns are able to go two squares and one square forward on the first turn and then one square forward from that point onwards. The pawns also have a special move called en passant. This is where if your pawn is has moved three rows ahead of its home square and your opponent moves a pawn two squares forward such that that pawn is adjacent to your own pawn, it is able to diagonally capture the pawn even though they are diagonally facing each other (the pawns will be facing each other laterally). The knights move in an L-shape following the vector (1,2) in various permutations. The bishops are able to move through open diagonals to anywhere along a specific diagonal until there is a piece blocking their path. If it’s one of your own pieces (i.e. if you have the white pieces and you have one of your own pawns blocking the path), you shouldn’t be able to complete a capture, replacing said captured piece on that square. However, if it’s an enemy piece, you should be able to complete the capture and there will be a small collection of all of the pieces that have been captured in the game by either side. The rooks move up and down and are also involved in the special move called castling, which I will get onto later. The queen combines the functionality of a bishop and a rook, and so I have been able to re-use those functions again to create the queen moves functionality. The final piece is the king, which is able to move one square in every direction and it is able to do this. When the king is attacked, the king is put in check, which I have demonstrated to the user through making the king flash red. In the normal game of chess, the side whose king is in check has to either move the king out of check, block the check with another piece or capture the offending piece. However, as I am demonstrating, I wasn’t able to implement this functionality, so I chose to make it clearer by making the king flash. Additionally, while this functionality could have been implemented, I realised that my end-client was a relatively experienced club player and that he would be able to spot this through a simple prompt like making the king flash. Upon conversing with my end-user, I was able to confirm this suspicion, and this meant that I could then move onto implementing other components of the project. In addition to this, the king also has a move called castling, which is the only instance where the king is able to move two spaces. If the rook to which side the king is moving hasn’t moved in the game yet and there aren’t any pieces that are currently blocking the move from taking place, then the move can be made. There is also the additional constraint of whether the king hasn’t moved as well. If these conditions are met, then the move is legal and will be drawn to the user upon selecting the piece. The only difference between this game I’ve implemented and the actual game itself is that you are still able to castle through check as I will demonstrate, which is illegal in the actual game.

Look at promotion.

Talk about clicking off the board.

Clicking on a square that isn’t legal for a given piece.