Ajay Tadinada (at663), Harrison Sremac (hcs59), Mericel Tao (mst223), Sanya Kohli (sk2682)

Test Plan

Our team approached testing using a glass box approach, meaning we were very specific about how we wrote each of our test cases. Our reason for choosing this approach was that our implementation was done in a certain way and we wanted to push the boundaries of how we implemented our functionality. Furthermore, because chess is a game that has very specific and intricate rules, our team decided it made the most sense to make manual tests to check all of the functionality of our game.

Thus, all the parts of our system were tested manually by creating OUnit tests. We tested modules Board and Pieces using OUnit. As mentioned earlier, our test cases were developed using glass box testing. This was because we wanted to ensure compatibility with our code. Furthermore, we used bisect in order to ensure that our lines were covered.

To start testing, we tested initialization of the board to make sure pieces were where they should be according to the rules of chess.

After that, our approach to testing began by looking at each of the individual pieces. As such, we created tests for each correct movement for the type of piece (pawn, knight, rook, bishop, queen, and king). To check the other side of this, we also made test cases to confirm that incorrect movements for these pieces were not allowed. In addition to that, we wanted to make sure that pieces were not allowed to move off the board, so tests were created for that. We decided to think of other possible moves and made test cases for those as well including captures and special chess moves. Testing these movements was very important and checked a lot of our functionality.

In addition to movement, we checked for the winning positions, confirming that checkmate and stalemate are true and false for the correct game boards.

Because one of the biggest rules with chess is how each piece can move. After testing the movement of each piece to confirm they move correctly, we were very confident in the correctness of our system. This, combined without other test cases like checking board statuses and outputs, gave us confidence that our test cases demonstrated correctness. Furthermore, our bisect coverage was high (92% for board.ml and 94% for pieces.ml), reassuring us that we implemented tests in an efficient way.