Test Plan
Runs with dune test

Automatic Testing

1. We used OUnit testing to ensure correct functionality of our chess game. All of the OUnit tests were made for the backend Movement module. This module includes all of the backend logic for the game such as piece movements, checks, captures, checkmates, draws, and other important game rules.
2. Automated tests covered:
    - Piece Movement Validation: Ensuring that each chess piece (Pawn, Rook, Knight, Bishop, Queen) moves according to chess rules.
    - Game State Evaluation: Checking for end of game states (checkmate or different draws)
    - Special Rules: Checking special movement of pieces such as en passant and castling.
    - Utility Functions: Testing helper functions that determine piece color, piece position, and player turn.
3. We wanted most of the tests to be black box tests, designed to ensure the code followed the rules of chess without worrying as much about how it was implemented. This testing strategy was to ensure that the chess game worked properly, focusing more on the rules of the game than how it was implemented. In reality, we had to know more about the backend to get the tests to work because of how our functions interacted with the GUI. So our testing was a mix of black box and glass box. We also focused on trying to test a good amount of edge cases, a blackbox testing strategy to ensure certain rules of the game were upheld (like trickier rules such as en passant and draws). Since there are so many different edge cases in the game of chess we could not check them all. However, we did make sure that basic cases of each rule of the game work such as basic piece movement, obvious check and checkmate scenarios, etc.

Manual Testing

We did a large amount of our testing manually since our chess game is completely GUI based. While we were implementing functions, we would always manually test them along with creating automatic tests for them. Without the manual tests we would not be able to ensure that the chess game actually worked as we expected it to. Because manual testing is also easier, we were able to test more edge cases on manual testing to ensure that no unexpected behavior occurred while playing the game. The manual testing consisted of simply playing the game of chess by clicking on different squares to move the pieces. By playing out different moves of the game we tested to ensure that the GUI accurately showed what we expected it to, ensuring that piece movement and game rules were implemented properly. We also played out some entire games to ensure that the game worked as intended. Overall, the frontend (GUI) was completely manually tested, but all of the backend was also tested manually at some point to ensure that the game behaved as expected.

Module tested and how the test cases were developed

The testing primarily targeted the Movement module (movement.ml), where all of our backend code resides. The comprehensive testing of this module ensures that all possible movements and game states are correctly implemented and interact as expected.

Justification for Testing Approach

Our testing approach demonstrates the correctness of our game by focusing on making sure the rules of the game are not violated. The automatic tests are to help ensure that basic implementation of functions work properly as well as some difficult edge cases. These tests also guarantee that some of the helper functions work properly, helping to guarantee that the parent functions will also work as expected. The manual testing is the main way we demonstrate the correctness of our game since this is what a user will experience. By manually playing through numerous positions and edge cases, we help to ensure that the game works as intended under all conditions. Since we manually tested the game a large number of times without any errors, it is highly unlikely that the user will encounter an error. This combined with the automatic testing, almost certainly guarantees that there are no bugs with the basic implementation as well as the edge cases specifically covered in automatic testing. By focusing on testing the rules of the game, we have guaranteed that the game can be played through without errors and will eventually reach an end state, exactly as the normal game of chess would.