# Chess Test Plan

**Manual vs. OUnit Testing:**

We noticed that much of manual testing and OUnit testing ended up overlapping.

One instance was while testing valid_moves and all_moves. We used manual testing as integration testing to ensure that the backend and frontend worked together (e.g. with these methods, piece moves were restricted on screen, which was displayed by highlighting places where a piece could go and not allowing the piece to move outside the region). Written physical tests covered edge cases and specific scenarios with only a few pieces (e.g., only a King and a Pawn, for example). This demonstrates the correctness of moves as it covers extreme cases through automated testing, and physical testing ensures that the system for moves operates seamlessly (without lag or inaccuracy in movements on screen) and pieces are able to interact with a multitude of other pieces on the board (i.e. taking erases the other piece). The tests for these functions were developed mainly through black box testing as the edge cases we ran into were either found while doing manual testing (while playing chess on the GUI) to simulate real chess scenarios and thinking through various real chess scenarios ourselves rather than going through the code itself to figure out what to test.

Everything except display and bitmap_file compilation units were OUnit tested. The display and bitmap_file modules were mainly manually tested since they involved user interaction and graphics. Though, while testing the other modules, we were able to achieve ample coverage of them (Bisect data: bitmap_file - **92%**, display - **65%**).

**OUnit Testing:**

We split up our OUnit testing into 3 files that tested different modules:

- test_game.ml
- test_matrix.ml
- test_position.ml
  *The piece compilation unit is inherently tested by the test_game file. It would be redundant to test it separately because the test_game.ml tests setting up and moving the pieces. Moreover, with these tests, piece.ml got **96%** Bisect coverage.

**test_srk_final_3110.ml: runs test_game.ml, test_matrix, and test_position in a single test suite

**test_game.ml:**

- Testing Valid Moves - Black Box Testing to ensure testing of real chess scenarios
  - valid_moves underwent many tests since it is the most critical function for the chess game since the application should not allow invalid moves to occur
  - Each kind of piece was tested thoroughly (i.e. all possible moves for knight, rook, queen, etc.)
  - Many chess scenarios were tested (i.e. stalemate, check, checkmate, castling scenarios, en passant scenarios, pawn promotion, etc.)
  - The detailed diagrams of our testing plan for this portion can be found at the bottom of this document
- Game state updates - Black Box Testing to ensure testing of real chess scenarios

        - Castling preconditions: moved.king, moved.right_rook, moved.left_rook
            - Ensured the game's move field updated if the king or left/right rook moved
        - Forward and Backward - Black Box Testing
            - Ensured previous game board was saved correctly without modification (since it is mutable)
            - Ensured future states were retained after moving backwards
    - player_move testing - Glass Box testing of valid and invalid user inputs
        - Mouse clicks and keyboard clicks were tested
        - Mouse clicks tested: same grid twice, select and then select another grid (valid move and invalid move)
        - Keyboard inputs tested: h (help), b (backward), f (forward), r (reset), q (quit), and bad input
        - Tested to see if expected game states resulted from the processed input
    - Verified initial placement of chess pieces - Black Box test
    - Bisect Coverage: **94%**, confidence in testing correctness

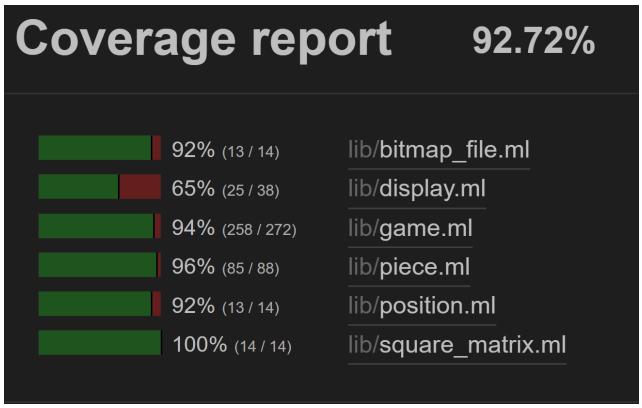
**test_matrix.ml:** Black Box Testing

- Tested all square_matrix module functionality with different primitive types of matrices to ensure primitive type compatibility with matrices.
- After writing the black box tests, a **100%** Bisect coverage of square_matrix.ml was observed, strengthening our confidence in the correctness of the implementation.

**test_position.ml:** Black Box Testing

- Tested all position module functionality, critically including constructing and moving a position.
    - Took into account niche cases, such as tests with and without optional arguments.
- After writing the black box tests, a **92%** Bisect coverage of position.ml was observed, strengthening our confidence in the correctness of the implementation.

**COVERAGE REPORT:** Using Bisect

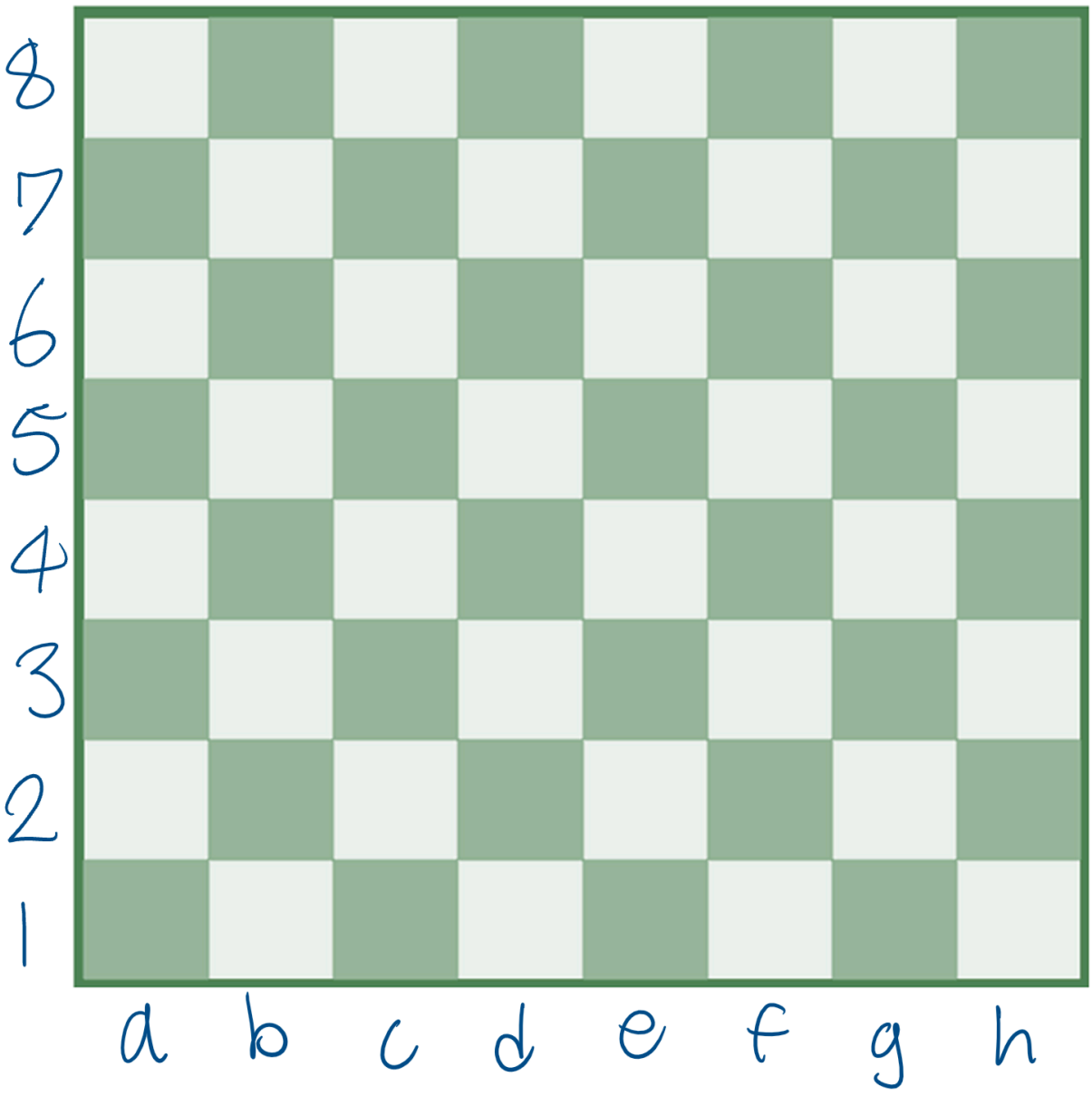


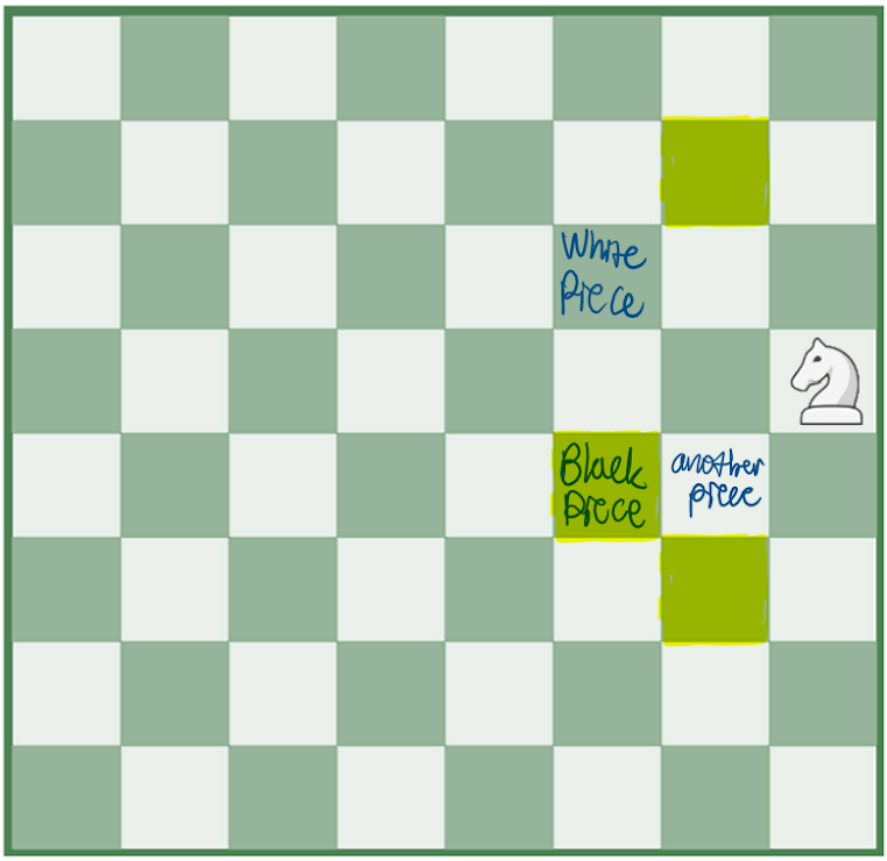

**CORRECTNESS OF OUR SYSTEM:**

In addition to the wide variety of test scenarios we covered as aforementioned, the coverage report gives us additional confidence in the correctness of our system since our coverage is **over 90%** overall and is high for the modules we tested automatically. Display has lower coverage due to next_input not getting

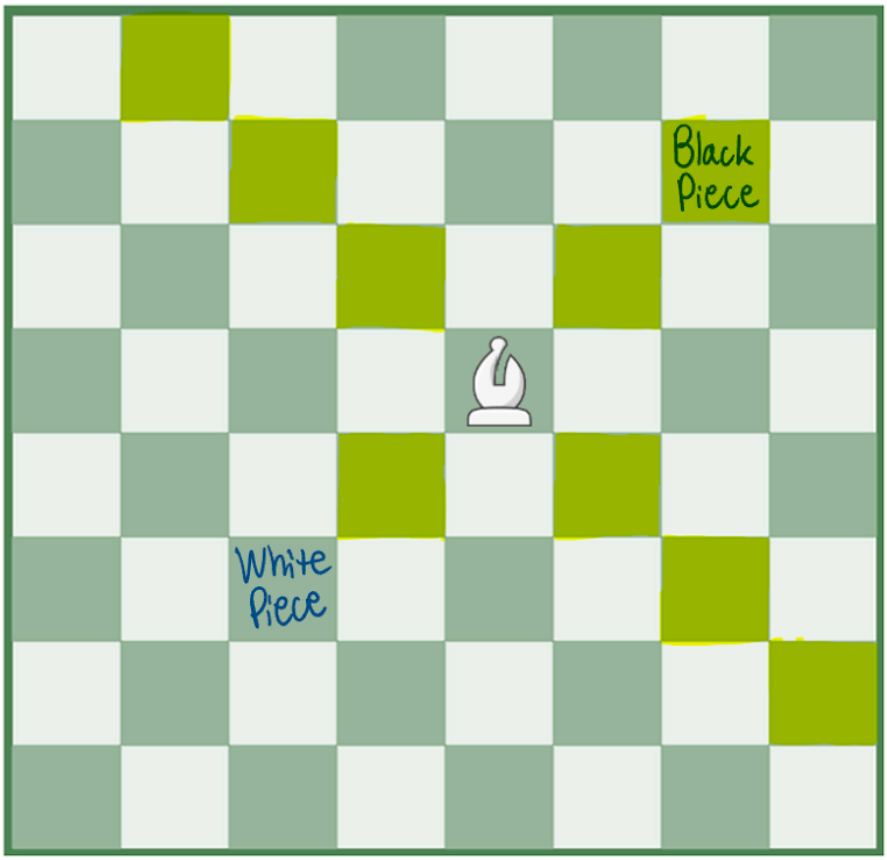
tested since waiting for user inputs is not able to be tested in OUnit. Note that piece.ml got ample coverage due to testing of the game module. Overall, we are confident in our testing approach and how it demonstrates the correctness of our system.

# Chess Test Case Illustrations

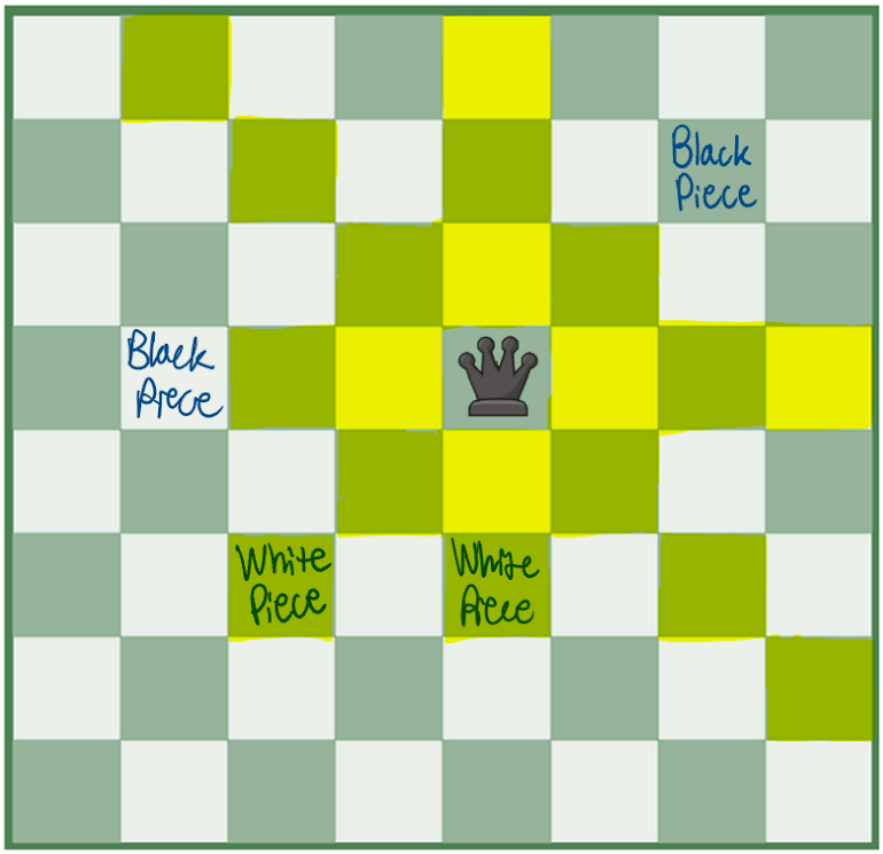




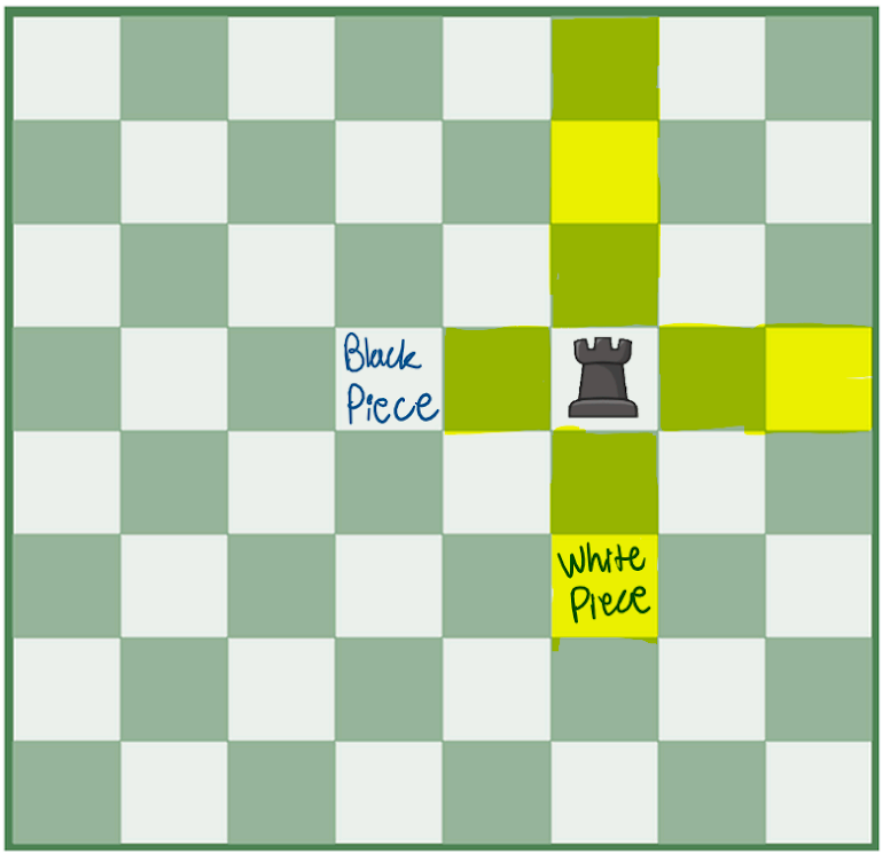

Knight testing board: accounts for jumping over pieces, not going off the board, not taking own pieces, and taking opponent pieces



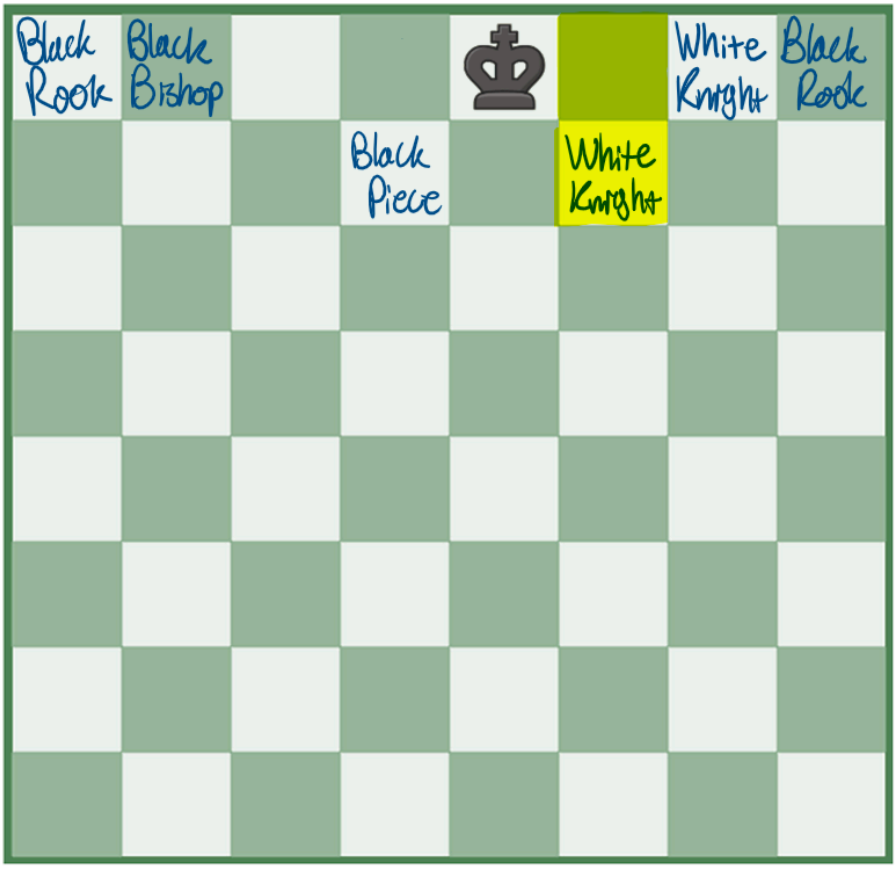

Bishop testing board: accounts for not jumping over pieces, not going off the board, not taking own pieces, and taking opponent pieces



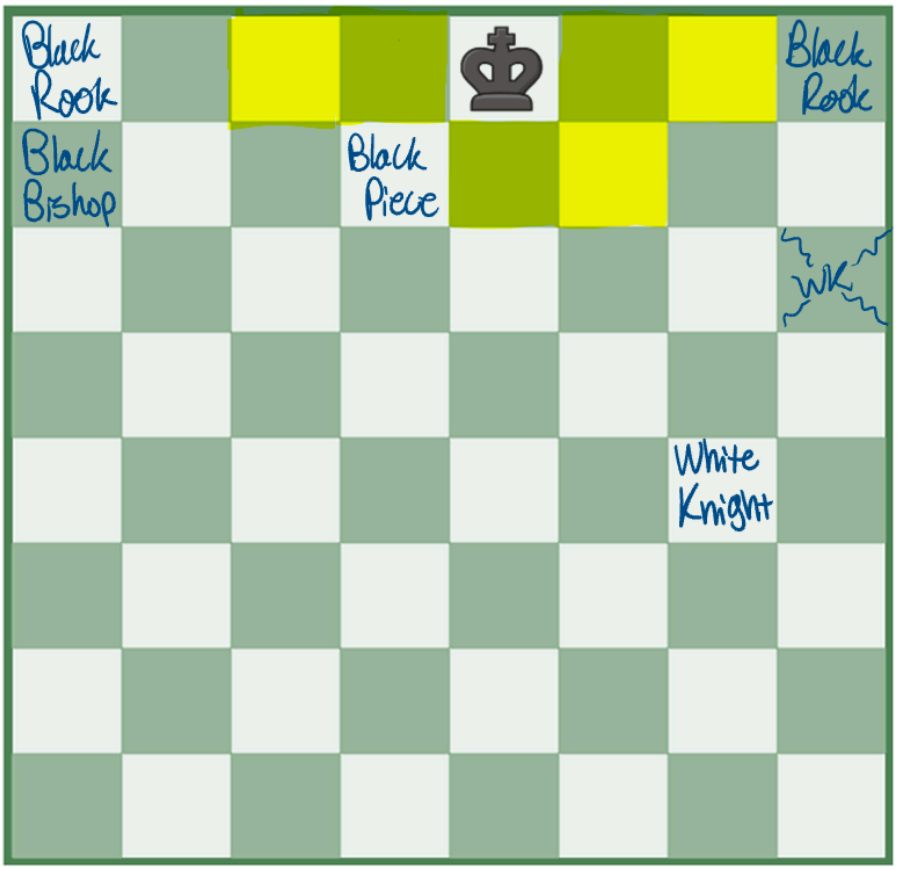

Queen testing board: accounts for not jumping over pieces, not going off the board, not taking own pieces, and taking opponent pieces



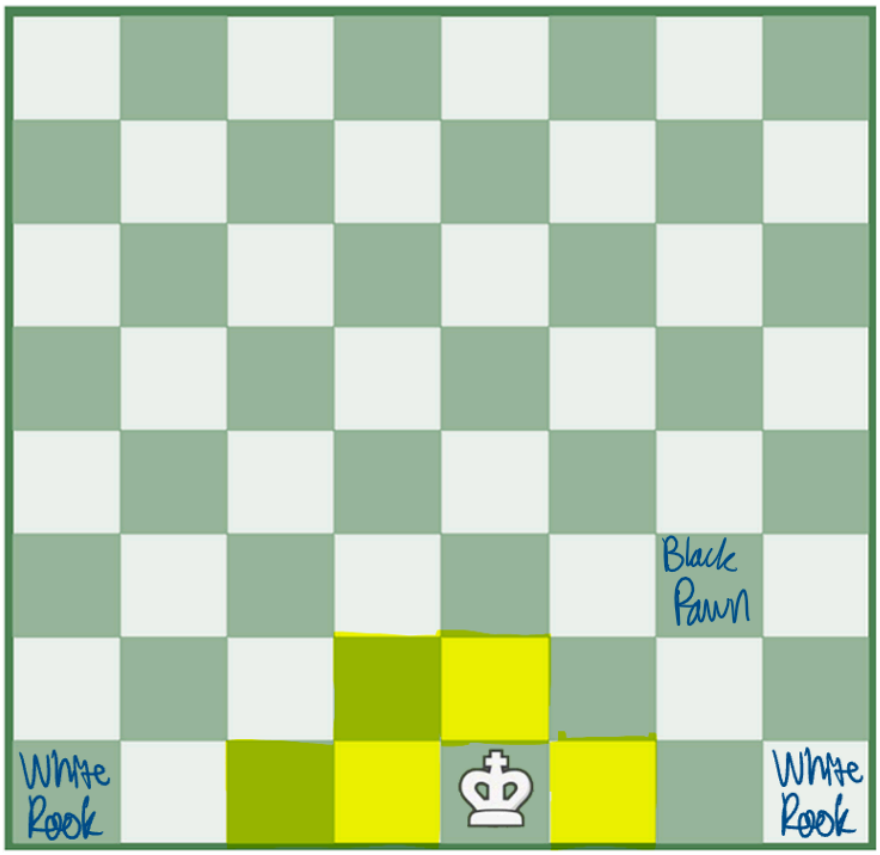

Rook testing board: accounts for not jumping over pieces, not going off the board, not taking own pieces, and taking opponent pieces



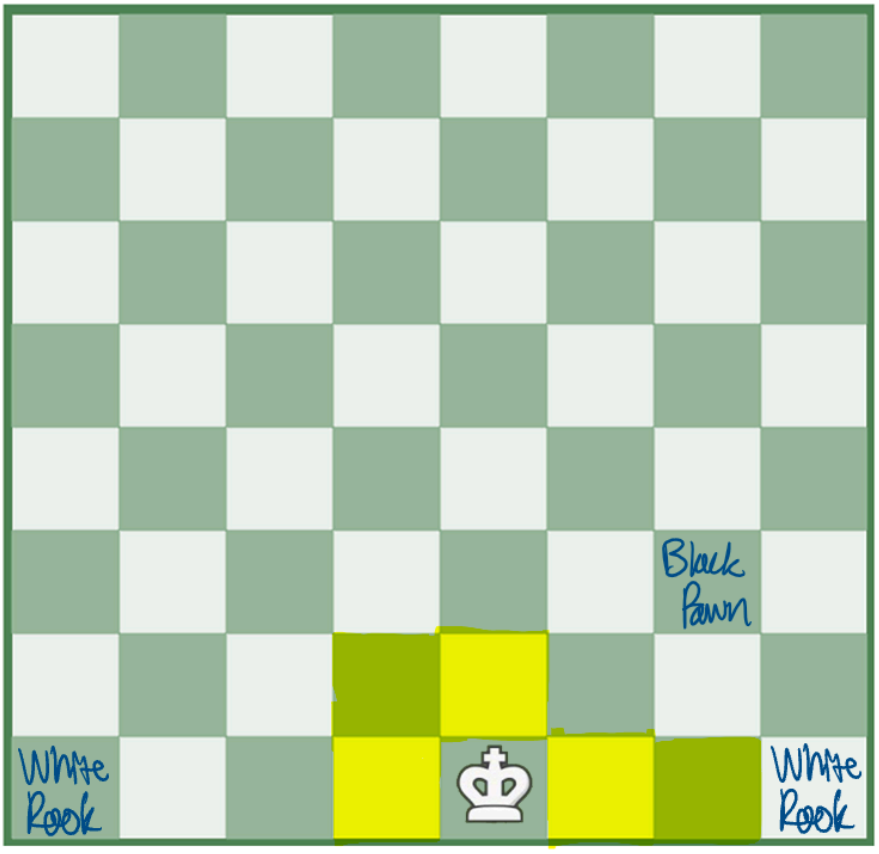

King testing board 1: accounts for no castling when path is blocked, not going off board, not taking own pieces, taking opponent pieces, and not going into check.



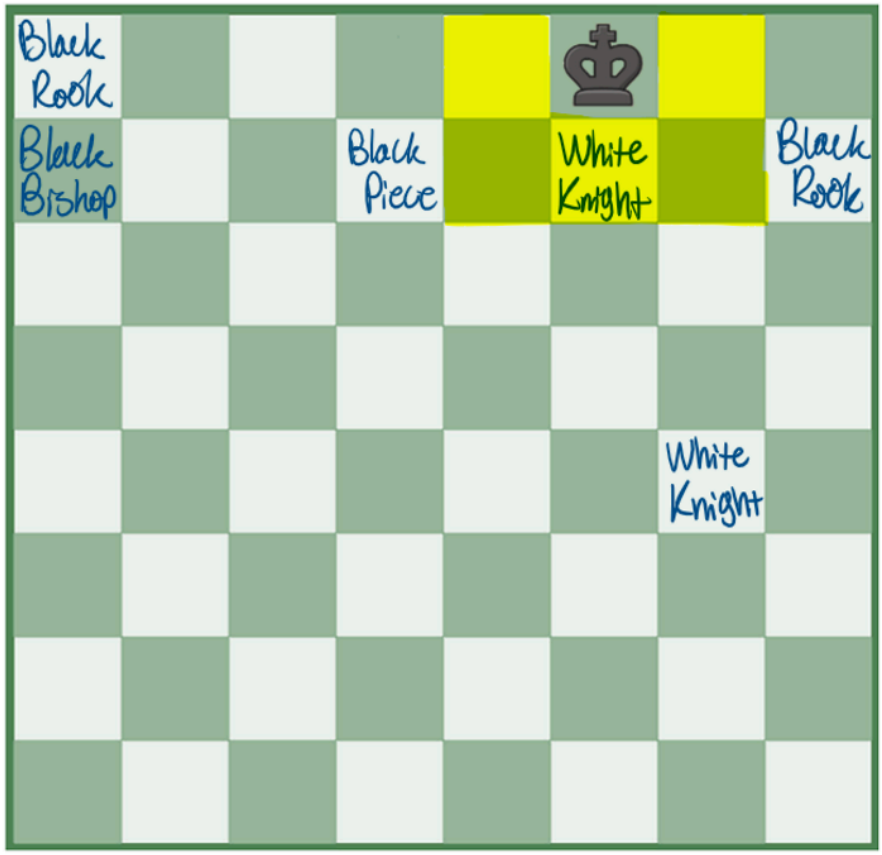

King testing board 2: moves the blocking pieces so all castling is now possible



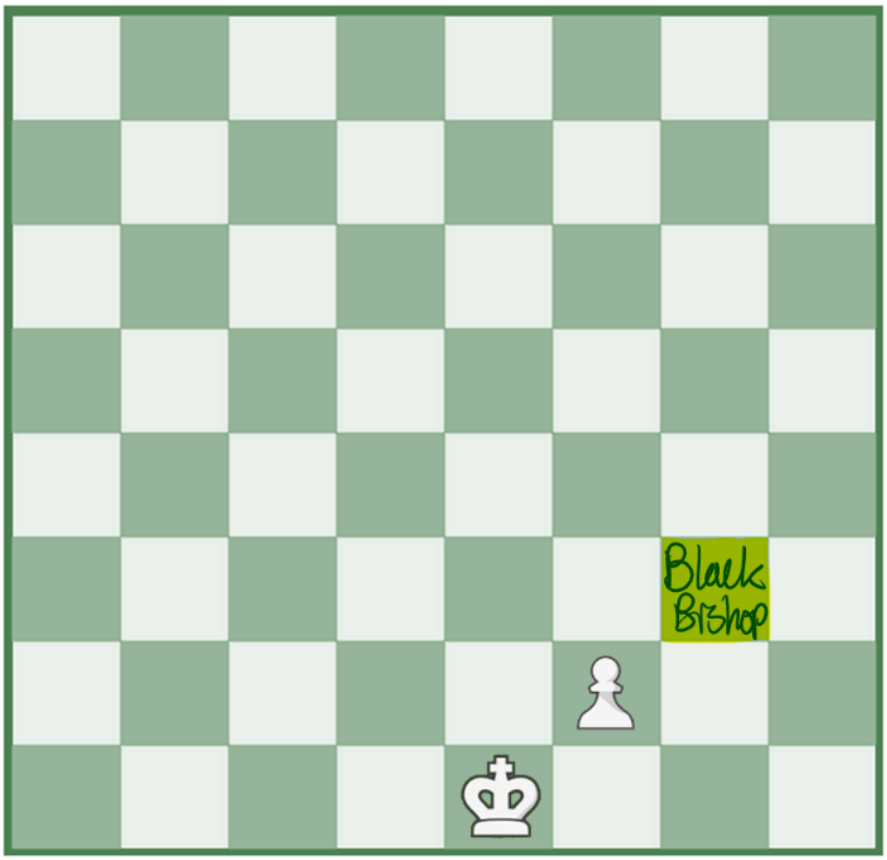

King testing board 3: regular castling is not possible since kingside rook has moved



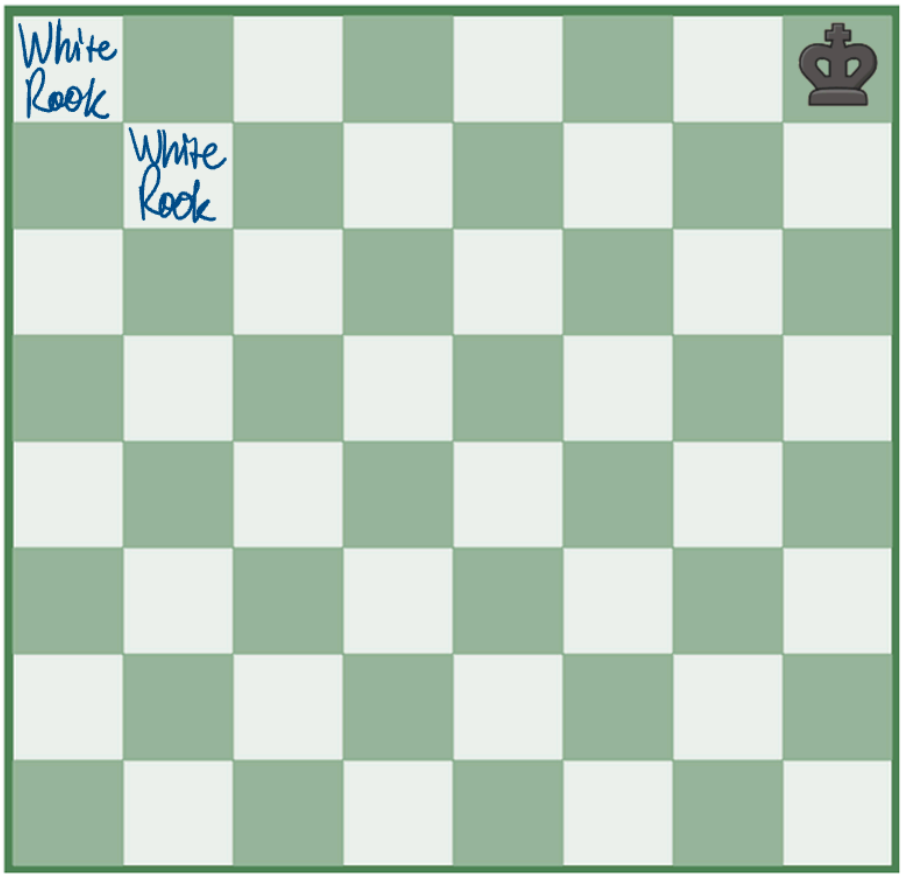

King testing board 4: long castling is not possible since queenside rook has moved



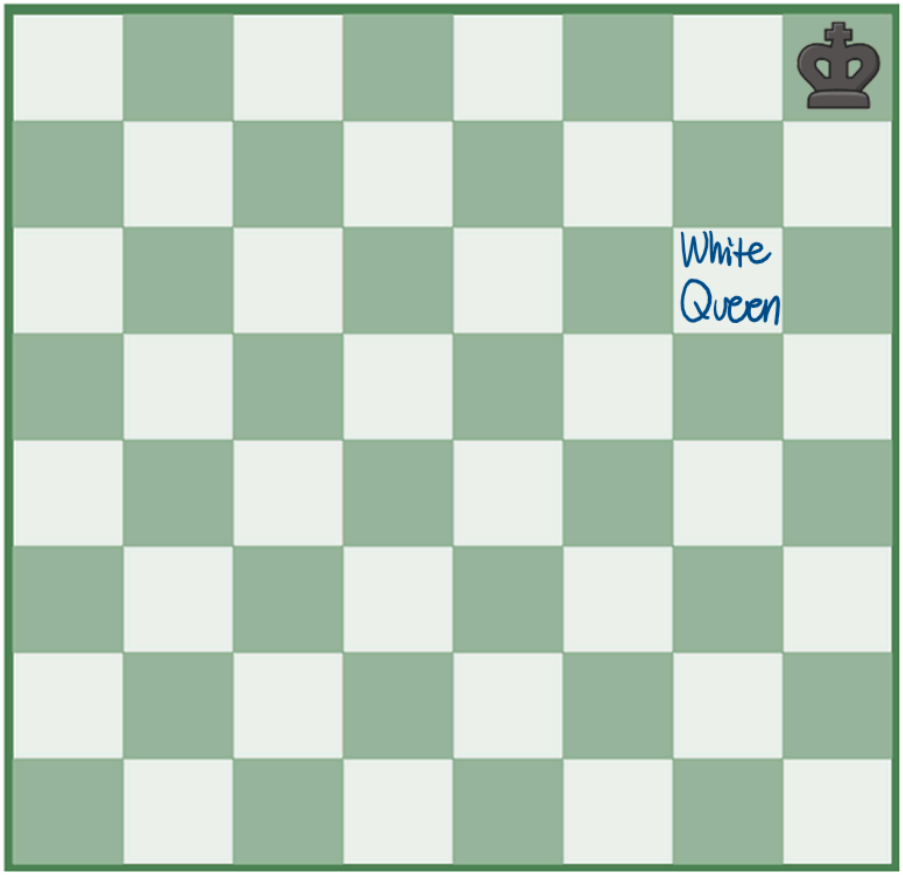

King testing board 5: the king moved so castling is no longer possible



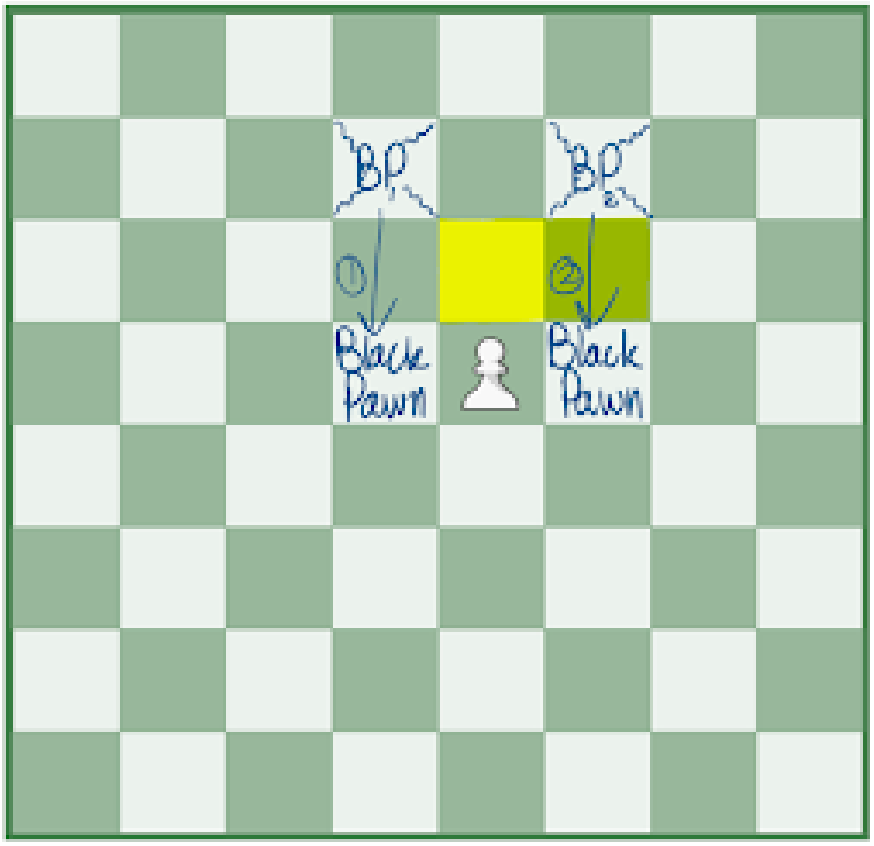

King testing board 6: Other pieces cannot move in such a way that would expose the king to attack but they can move to protect the king. In this case, the pawn can take the bishop but cannot move forwards



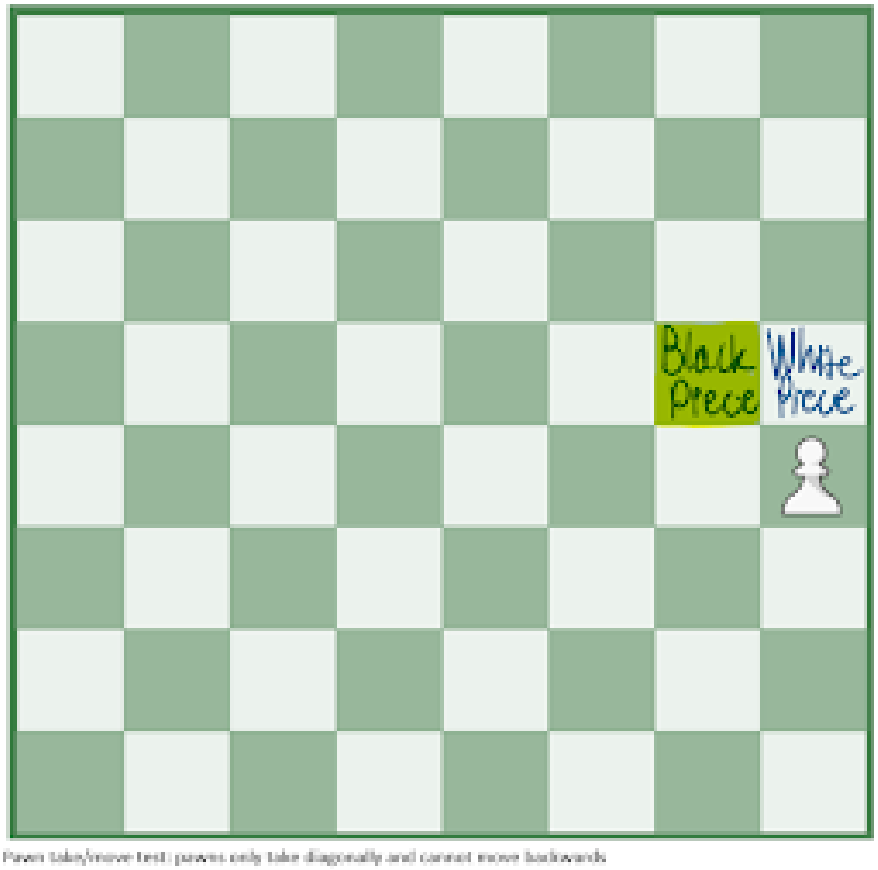

Checkmate test



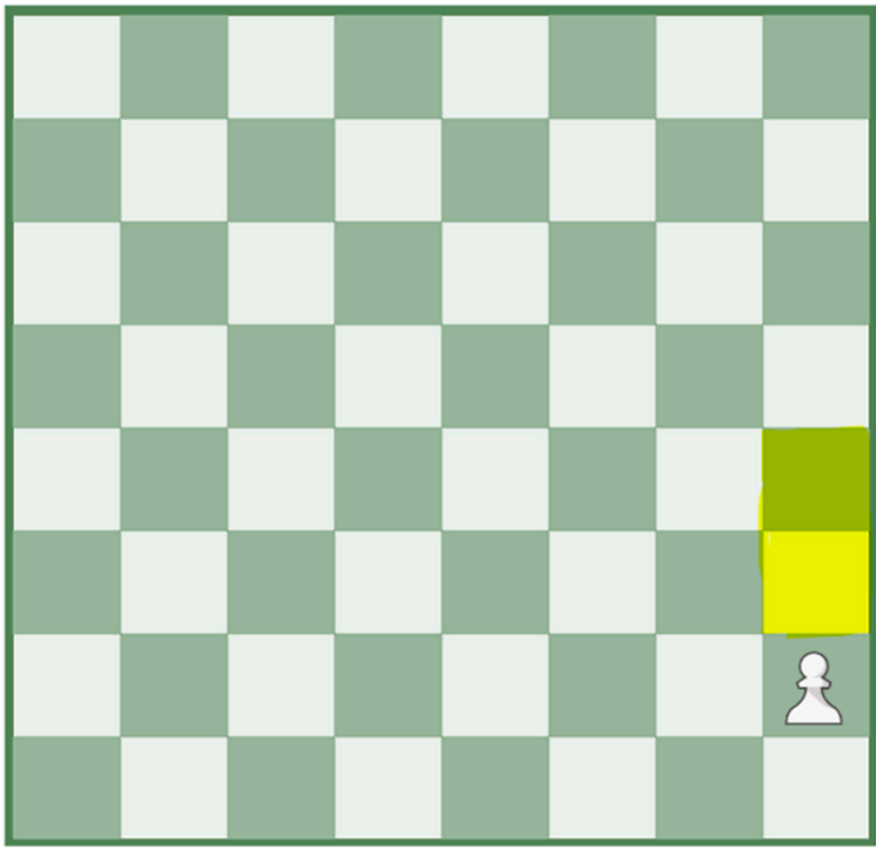

Stalemate test



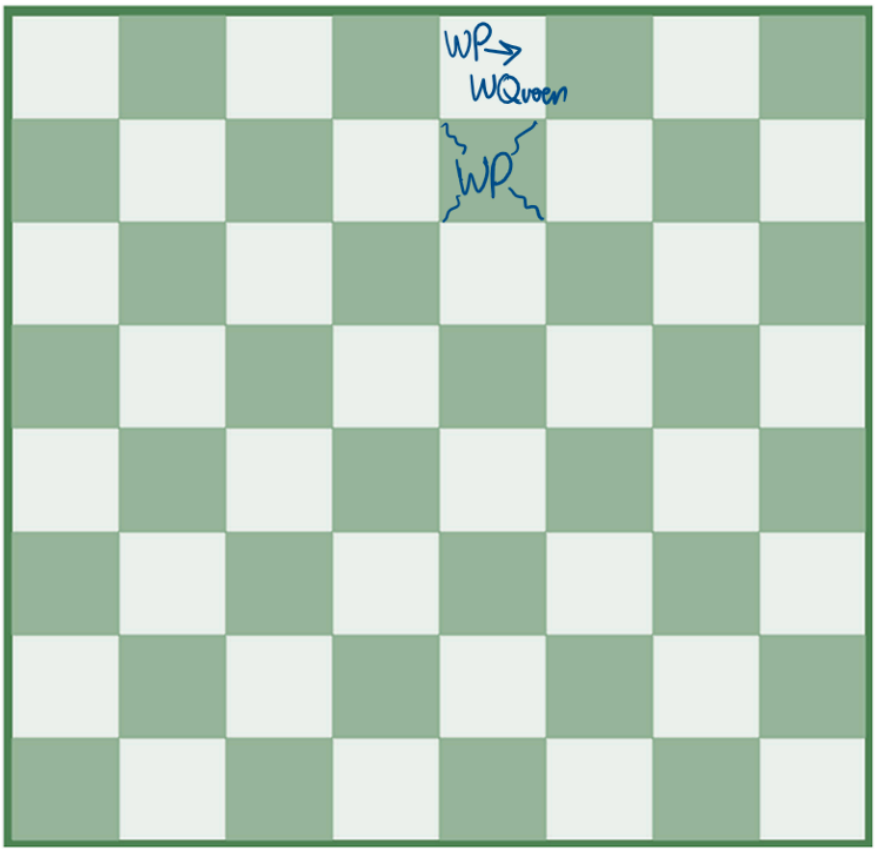

En passant test: only pawns that have just moved can be taken by en passant




Pawn take/move test: pawns only take diagonally and cannot move backwards



Pawn initial move test: pawn can move 1 or 2 squares in its first move.




Pawn Promotion: move white pawn to other edge, make sure it becomes a queen