| ID | Description | Steps | Expected | Actual | Results | Comment |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Initialization | Create a new instance of the Chess class. | The Chess object is initialized and a new game of chess is started using the starting position. | The Chess object is initialized with no error and starts at expected. | ✅ |  |
| 2 | Display | Call the display() method on the Chess object. | The current state of the chess board is displayed using chess characters. | The board is displayed very well and each piece is placed in their correct position. | ✅ |  |
| 3 | Check state | Call the check\_state() method on the Chess object after making several moves. | The check\_state() method returns a string indicating the current game state based on the current board configuration and move history. | This function returns the correct state of the game, this is very useful for a smooth game experience. | ✅ |  |
| 4 | Fifty move rule | Make no pawn moves or piece captures for 50 consecutive moves, then call the fifty\_move\_rule() method on the Chess object. | The fifty\_move\_rule() method detects that the fifty-move rule has been triggered and prompts the player to choose whether they want to claim a draw or not. | This is detected. | ✅ |  |
| 5 | Five-fold repetition rule | Reach a position that has occurred five times previously, then call the five\_fold\_rule() method on the Chess object. | The five\_fold\_rule() method detects that the game has reached a five-fold repetition and declares a draw. | This is detected. | ✅ |  |
| 6 | Checkmate | Reach a board state that is a checkmate, then call the is\_checkmate() method on the Chess object. | The is\_checkmate() method correctly determines that the game has ended in a checkmate. | Based on the state of the game and the position of each piece on the board, a checkmate is determined correctly. | ✅ |  |
| 7 | Dead position | Reach a board state that is a "dead position" where neither player can win, then call the is\_dead\_position() method on the Chess object. | The is\_dead\_position() method correctly determines that the position is a "dead position". | This is detected correctly. | ✅ |  |
| 8 | Draw | Reach a board state that is a draw, then call the is\_draw() method on the Chess object. | The is\_draw() method correctly determines that the game has ended in a draw. | The draw is detected correctly. | ✅ |  |
| 9 | Test \_on\_check | 1. Create an instance of the chess game. 2. Player the game to a position where a king is on check 3. Call the \_on\_check() method 4. Verify that is returns a tuple contain True as the first element 5. Verify that the second element in the return value are valid moves. | It correctly detects if the game is on check and returns the available moves. | The function works as expected and from the output we are able to detect if the check is a checkmate, a stalemate or a regular check | ✅ | This function was used to implement get\_board\_state() |
| 10 | Test that previous moves are stored | 1. Create an instance of the Chess game 2. Place pieces on the board 3. Make a single move 4. Check that the last\_move property on the instance hold all previous state of the board | Correctly stores the state of the board prior to each move so that the board can be reverted to a previous state | The class works as expected by taking in data from the game and storing in the linked list node. | ✅ | The functionality was used to implement so many other features such as undoing of moves and saving progress. |
| 11 | Saving and undoing previous moves |  | Correctly saves the game’s state in a json file and restores a previous version of the game when a player reverts his move. | This functionality works as expected. The whole game along with previous moves are serialized into json format and saved to a file. The data in the file can be loaded into the game to bring the game back to the exact state it was before. | ✅ | We faced many challenges when building this functionality mainly due to the same object being used in many places, this created a bit of confusion but we were able to resolve the issue by copying lists and dictionaries. |
| 13 | Test the move() method of ChessBoard class to ensure that a piece can be moved to a new square on the board. | 1. Create an instance of ChessBoard class. 2. Place a piece on the board. 3. Call the move() method of the instance with the current location and the new location of the piece. 4. Check that the piece has been moved to the new location on the board. | The piece should be moved to the new location on the board. | The piece has been moved to the new location on the board. | ✅ | The move() method of ChessBoard class successfully moves a piece to a new square on the board. |
| 14 | Test the get\_all\_moves() method of the game class to ensure it returns a dictionary of all possible moves for each piece on the board. | 1. Create an instance of the game class. 2. Set up a board state with multiple pieces of both players. 3. Call the get\_all\_moves() method. 4. Check that the returned value is a dictionary. 5. Check that the number of keys in the dictionary is equal to the number of pieces on the board belonging to the current player. 6. Check that each key in the dictionary represents a piece on the board belonging to the current player. 7. Check that each value in the dictionary is a list of valid moves for the corresponding piece. | * The get\_all\_moves() method returns a dictionary. * The number of keys in the dictionary is equal to the number of pieces on the board belonging to the current player. * Each key in the dictionary represents a piece on the board belonging to the current player. * Each value in the dictionary is a list of valid moves for the corresponding piece. | * The get\_all\_moves() method returns a dictionary. * The number of keys in the dictionary is equal to the number of pieces on the board belonging to the current player. * Each key in the dictionary represents a piece on the board belonging to the current player. * Each value in the dictionary is a list of valid moves for the corresponding piece. | ✅ | get\_all\_moves() method of the game class returns a dictionary of all possible moves for each piece on the board as expected. |
| 15 |  | 1. Create an instance of Chess class with a new game 2. Call the get\_game\_state() method to get the initial game state 3. Move a piece to put the opponent king in check 4. Call the get\_game\_state() method to get the game state after the move 5. Move the opponent king to a safe square 6. Call the get\_game\_state() method to get the game state after the move 7. Move pieces to a stalemate position 8. Call the get\_game\_state() method to get the game state after the moves | * Tuple containing False for is\_check, False for is\_checkmate, False for is\_stalemate, and empty list for c\_escape * Step 4: Tuple containing True for is\_check, False for is\_checkmate, False for is\_stalemate, and a list of safe squares for the opponent king for c\_escape * Step 6: Tuple containing False for is\_check, False for is\_checkmate, True for is\_stalemate, and empty list for c\_escape * Step 8: Tuple containing False for is\_check, False for is\_checkmate, True for is\_stalemate, and empty list for c\_escape | * Tuple containing False for is\_check, False for is\_checkmate, False for is\_stalemate, and empty list for c\_escape * Tuple containing True for is\_check, False for is\_checkmate, False for is\_stalemate, and a list of safe squares for the opponent king for c\_escape * Tuple containing False for is\_check, False for is\_checkmate, True for is\_stalemate, and empty list for c\_escape * Tuple containing False for is\_check, False for is\_checkmate, True for is\_stalemate, and empty list for c\_escape |  |  |
|  |  |  |  |  |  |  |