CS 246 Final Project Chess - Demo

Yuktesh Modgil (ymodgil), Reyansh Patange (apatange), Harsh Dave (h7dave)

Our group has chosen to create a Chess game for our CS 246 Final Project. This demo document will take you on a walkthrough of our program and show you various features that we have implemented.

To get started, please download the ZIP file containing our chess project and open it using your SSH client in the CS Undergraduate Environment. Once this is complete, please type the command "make" in your terminal to compile the program into an executable called chess. Now, you are ready to take a tour of our program and all of its features.

Game

Let's begin with the game command, which is entered as "game *white-player black-player*", where *white-player* and *black-player* are parameters that can be either *human* or *computer[1-4]*.

(Note: Since the graphical observer takes some time to render and computer1 makes random moves, you may want to test these last)

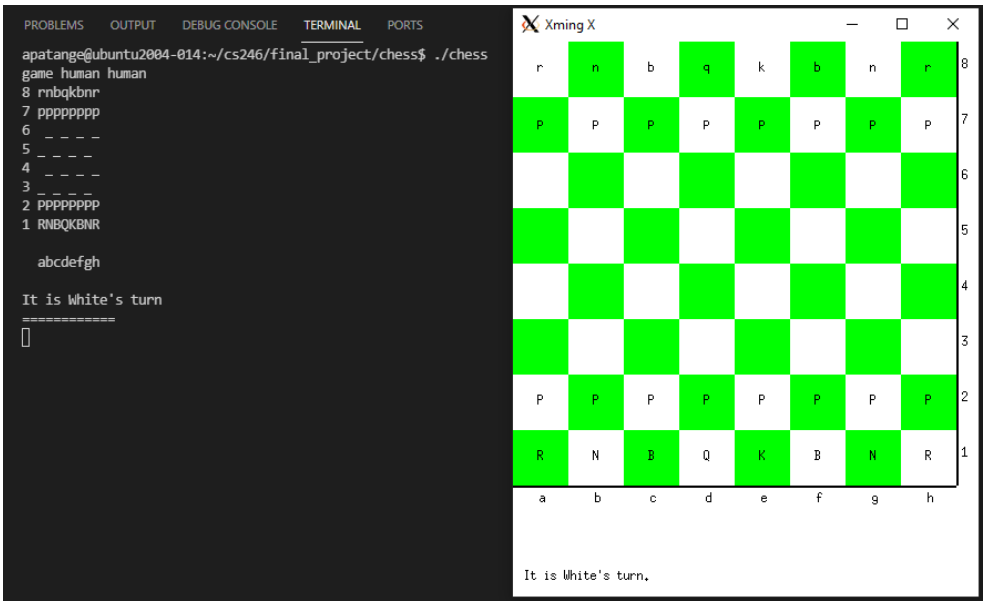
If you try running chess with *testgame1.in,* and *testgame2.in* you can see the result of running the "game human human" and "game computer1 computer1" commands. It will display the chessboard in the standard starting position and print out "It is White's turn" in the text and graphical displays. If you wish, you may play games between humans and computers using the command "game human computer[1-4]" or vice versa. We cannot provide a test for this, as the moves of the computer are unpredictable.

Next, you can test the other 3 levels of the computer by running the chess executable with *testComp2.in, testComp3.in,* and *testComp4.in* respectively. Level 2 prefers capturing moves and checks over other moves, level 3 prefers avoiding capture, capturing moves, and checks, and level 4 adds priority to moves instead of using random preferences like levels 2 and 3. Level 4 gives the highest priority to checkmate moves and check moves without capture. Then it decides to either kill or avoid capture depending on which move would provide more points, as each piece is assigned standard chess points (see below).

King	Queen	Rook	Knight	Bishop	Pawn
1000 pts	9 pts	5 pts	3 pts	3 pts	1 pt

Display: For the text display, capital letters are white pieces, lowercase letters are black pieces, underscores are black squares, and blanks are white squares, as specified on the chess project document. For the graphical display, capital letters are still white pieces and lowercase letters are still black pieces but the black and white squares are actually drawn on the graphical display window.

If the game command is given without entering setup mode, it will show the chessboard in the standard chess starting configuration in the text display and graphical display as shown below:



Setup

- Use *testSetup.in* to verify that setup is functioning and that pieces can be added and removed from the board. Entering "done" exits setup mode if the board is valid (i.e. the board contains exactly one white king and exactly one black king; that no pawns are on the first or last row of the board; and that neither king is in check)
    - The starting player can be set using "=" followed by the *black* or *white*.
    - Pieces can be added using "+" followed by the piece's character representation and the tile coordinate you wish to place it at.
    - Pieces can be removed by using "-" followed by the coordinate of the tile that contains the piece you wish to remove.
- Note: the *done* command must be issued to leave setup mode.

Move

<u>King</u>:

- Use *testKingMoves.in* to verify that the King moves one square in any of the 8 directions and shows the King's ability to capture pieces as long as it does not put itself in check. This test case also shows that invalid moves are not allowed (ex. moving out of bounds and moving to a square blocked by a friendly piece).
- Use *testKingCastle.in* to verify that the castling functionality is working as expected. The king moves two squares towards one of the rooks, and that rook then occupies the square "skipped over" by the king in one move.
- Use *testKingCastleInvalid1.in* to verify that that castling only works when the king and rook have not previously been moved in the game.
- Use *testKingCastleInvalid2.in* to test that there must be no pieces between the king and rook used.
- Use *testKingCastleInvalid3.in* to test that the king must not be in check on either its starting position, its final position, or the position in between (to be occupied by the rook).

Queen:

- Use *testQueenMoves.in* to verify that the Queen moves any distance in any of the 8 directions and shows the Queen's ability to capture pieces that lie in its path. This test case also shows that invalid moves are not allowed (ex. moving along a path blocked by a friendly piece).

Bishop:

- Use *testBishopMoves.in* to verify that the Bishop moves any distance only in the 4 diagonal directions and shows the Bishop's ability to capture pieces that lie in its path. This test case also shows that invalid moves are not allowed (ex. moving along a path blocked by a friendly piece).

Rook:

- Use *testRookMoves.in* to verify that the Rook moves any distance only in the 4 vertical/horizontal directions and shows the Rook's ability to capture pieces that lie in its path. This test case also shows that invalid moves are not allowed (ex. moving along a path blocked by a friendly piece).

Knight:

- Use *testKnightMoves.in* to verify that the Knight moves in directions (x ±2, y ±1) or (x ±1, y ±2) where (x, y) represents its current coordinates, and shows the Knight's ability to capture pieces that squares it can move to. This test case also shows that invalid moves are not allowed.

Pawn:

- Use *testPawnMoves.in* to verify that the Pawn moves 2 spaces forward on its first move and 1 space forward after that. The test also shows the Pawn's ability to capture pieces that lie 1 square diagonally to the left or right of it. This test case also shows that invalid moves are not allowed.
- Use *testPawnPassant.in* to verify that the Pawn can do an En Passant as expected.
- Use *testPawnPromotion.in* to verify that the Pawn can do a Pawn Promotion as expected.

Stalemate:

- Use *testStalemate.in* to verify that stalemate is functioning and the game is declared a draw when any player ever has no legal moves available but is not in check.

Check:

- Use *testKingCheck.in* to verify that the King is not allowed to put itself in check and the behaviour when either King is put in a check condition.

Checkmate:

- Use *testKingCheckmate.in* to verify that the current game ends if there is a checkmate, and the player that checkmates the opposite player gets a point.

Resign

- The resign command is demonstrated in all of the movement tests mentioned above to end the game prematurely, before a checkmate or a stalemate occurs.

Scoring

- Each of the above tests demonstrates the score of the players being shown once End-Of-File has been encountered.