CS 154 Project

Chess

(LazyBoy)

Navin Chandak # 110050047

Ayush Kanodia: # 110050049

Problem Description

The problem to be solved can be enumerated as follows.

Given an opponent a game of chess, attempt to play and beat the opponent at the game

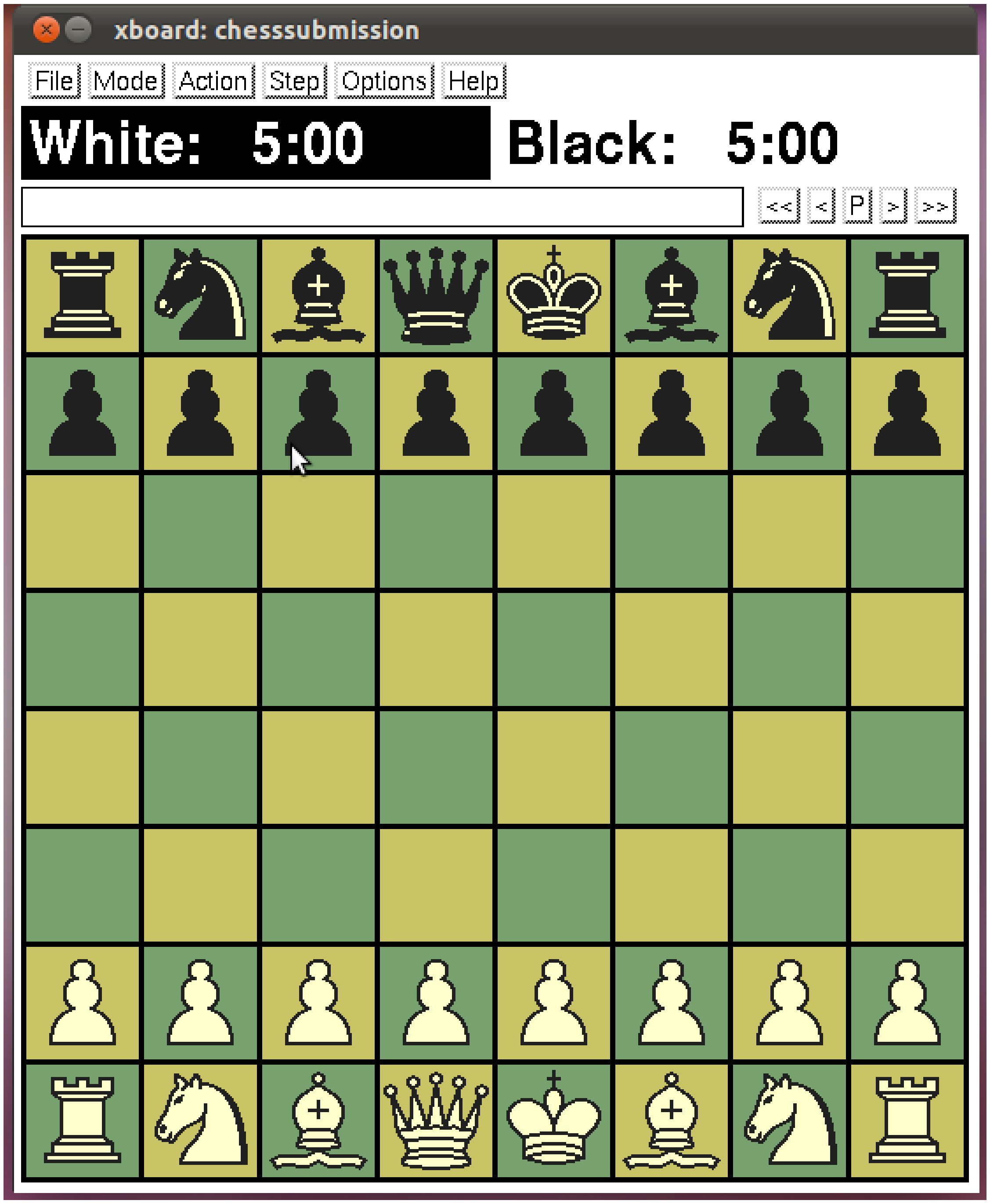
Simply put, the program plays a game of chess and employs artificial intelligence to describe which moves for it are the best to play at any given point in time.

Overall idea of the design of the program

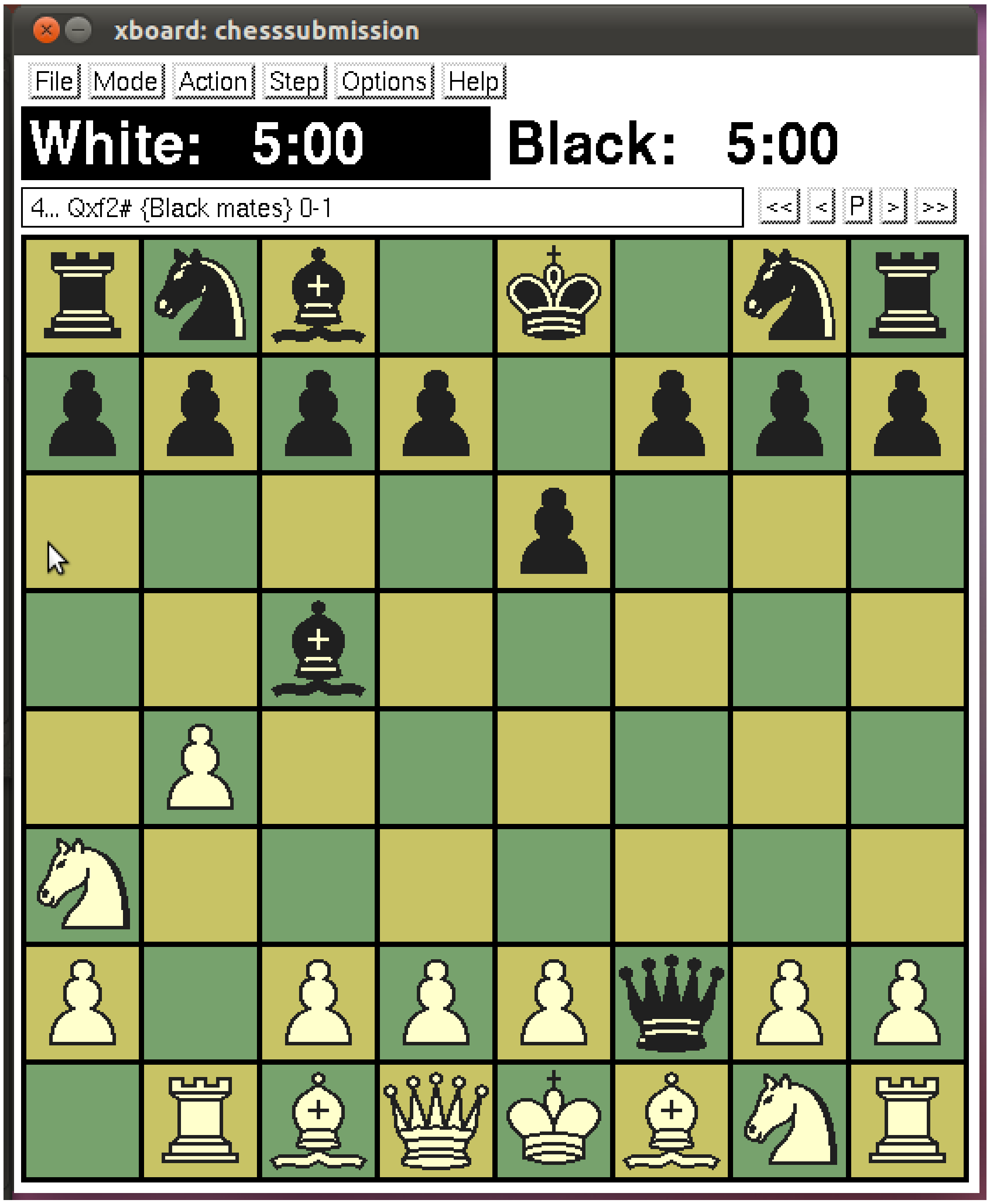
The problem of chess can be broken down into three sub-problems

1. Board representation
2. Listing of moves
3. Playing a move

Board representation, Chess rules, Xboard integration and Debugging



The code represents the board in a 2d-vector form. Then there are various functions which effect many changes in the board. Any changes to the board is stored in a list so that any move that has been done can be undone.We wanted to use a circular queue for this but realized that it won't be of much help because usually even a good chess game gets over in around 100 moves. Then the code includes a board display function which makes debugging easy. The code includes the function for getting the input from the xboard, parsing the string-information from xboard, and corresponding actions on my board as well as any other actions that need to be taken. A major part of this code includes interpreting the forsythe-edward-notation which is shorthand for storing chess-board state. This was important for saving games, editing chess-board position and was very critical in easy debugging. Whenever required, we check if the player or the chess engine has been defeated or if stale-mate has taken place and give this information to Xboard.



4... Qxf2# {Black mates} 0-1

Listing of moves

The code for finding out valid moves is pretty straight-forward. We have just ensured that the rules of chess are followed while finding out valid-moves. Many abstractions have been used to make the code shorter particularly in this part. All valid-moves including en-passant and castling have been incorporated.

Playing moves

The best move at any point in time is decided using the minimax algorithm, along with alpha-beta pruning. The heart of the artificial intelligence lies in the heuristic static chessboard evaluator. This is easily the longest part of the code. Interestingly, it is completely detached and can be treated as a separate entity from the rest of the code. Also, what is very interesting is that this evaluation function can be tweaked subtly to produce minor variations in results. Or, it may be tweaked morbidly to make the BOT lose!

The various features of the heuristic evaluation function have been listed in the code as comments, and are self-explanatory. There are about 35 conditions which have been incorporated. Each of these can be very simply tweaked by varying the number associated which each of these moves.

LIMITATIONS AND BUGS IN THE PROGRAM.

Bugs

Although we have tried our level best to test for the possibility of all kinds of bugs in the program, we are more than happy to concede the fact that unexpected bugs may still crop up at certain points in the game. These may be on two major counts, apart from other reasons. One reason is the usage of xboard, which may send unexpected responses to unencountered situations. The other is the artificial intelligence part, which may get confused at times. Other reasons such as a change in the number of moves available may create errors or unexpected responses, if the updations are not perfect. This however, should not happen as the code has been thoroughy tested.

Limitations

One major limitation is within the quality artificial intelligence in the game. This is because we do not have professional knowledge on the various game theoretic situations in chess, and neither has it been possible to incorporate all that we know. Also, for the same reason, certain responses of the artificial intelligence are not perfect, such as sluggishness in the beginning of the game, and repetitive moves during many situations of the game. Given the trade-off between time, the quality of the artificial move maker, and the quantity of features that we could have incorporated, we feel that we have found the best combination of the above ingredients. Please refer to the listing of moves section in order to find out the moves that have been included and those that have not been. The final limitation is the inclusion of all possible end game conditions, the various conditions for a draw and those for a victory. Conditions such as three fold repetition, the fifty move rule, and automatic draws due to not enough mating material available on the board, have been overlooked for now, and these may be included in future versions of the program, without much effort.

Point of interest: Future additions.

The best part of this project is its open ended nature, and the fact that this may be improved a long way further, given some more time. It has a sound base, the code written is hugely efficient (the demonstration of which will be given during the presentation) as well as self-explanatory, which makes it better for the code to be reused in the future. The artificial intelligence can be immensely improved, and this program can be made nearer to perfect.