**Artificial Intelligence (CSE3013)**

**Project Report**

B1 Slot : Professor Rajesh Kumar

*Robin Thomas 17BCE1330*

*Aadarsh 17BCE1233*

# An Artificial Intelligence Chess Program:

hess Board

**Introduction:**

*The goal of the project is to design and implement an artificial intelligence-based chess playing program. This is a widely famous problem that has been in the works for a very long time and various renditions of the problem are available in the form of chess engines and chess games easily available to download on the Internet. Since the advent of computers, people always viewed chess playing as a task suited for a computer.*

*The incident that made chess engines very famous happened way back in 1997 where the world champion at that time Garry Kasparov played a chess engine Deep Blue. During one of the game’s, he’s fidgeting in between turns and shaking his head in disbelief as he waits for his opponent to put the final touches on an inevitable victory. Finally, Kasparov makes his move, stands up and races away from the board. He raises his arms, astounded that he was beaten by a machine. His opponent was the IBM supercomputer Deep Blue, a machine that was capable of imagining an average of 200,000,000 positions per second.*

*So when Kasparov, one of the greatest chess players of all time, lost to a computer in front of a global audience, people began to wonder whether it was just a matter of time before machines surpassed humans in other aspects of life.*

*“From the beginning of the computer era, it was a belief that chess could serve as the ultimate test for machine intelligence,” Kasparov says. “And the game of chess has always been seen as the nexus for human intelligence. So, when a machine faced a human in chess and won this battle ... it could definitely be a revolutionary moment.”*

Figure

2 Garry Kasparov and the game of artificial intelligence

***Objective:***

*The goal of the project is to create a program that implements some of the basic known chess algorithms, allowing a user to play against the machine with a simple graphical user interface.*

***Working principle:***

*These are some of the basic concepts that will help us create a simple chess AI:*

*move-generation*

*board evaluation*

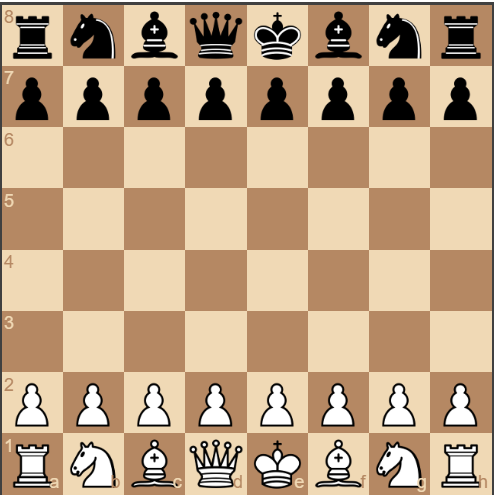
*minimax and alpha beta pruning. (Module 3 of CSE3013)*

*In addition to these concepts, we will also take advantage of certain libraries such as chess.js and chessboard.js to implement the rules of chess. Based on this, we can ensure that the chessboard ui is accurate and the computer makes only legal moves Finally, we will create the algorithm that finds the best move which is the ultimate goal of the project.*

***Design:***

### ***Move generation and board visualization:***

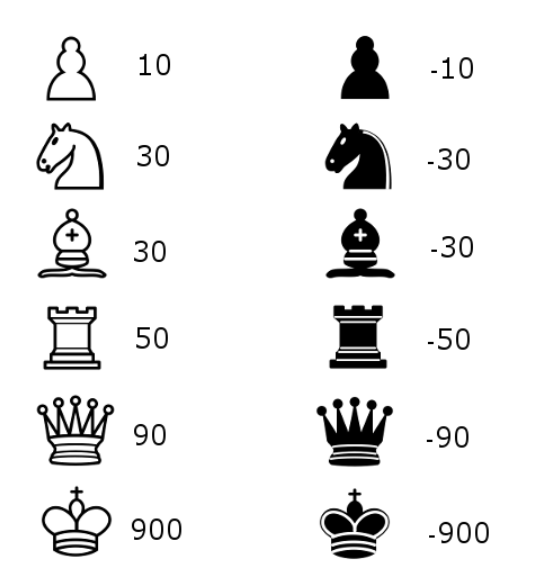
We’ll use the [chess.js](https://github.com/jhlywa/chess.js) library for move generation, and [chessboard.js](https://github.com/oakmac/chessboardjs/) for visualizing the board. The move generation library basically implements all the rules of chess. Based on this, we can calculate all legal moves for a given board state. Using these libraries will help us focus only on the most interesting task: creating the algorithm that finds the best move.



***Implementation details:***

1. ***Position evaluation:***

The simplest way to achieve this is to count the relative strength of the pieces on the board using the following table:



we’re now able to create an algorithm that chooses the move that gives the highest evaluation:

*var calculateBestMove = function (game) {*

*var newGameMoves = game.ugly\_moves();*

*var bestMove = null;*

*var bestValue = -9999;*

*for (var i = 0; i < newGameMoves.length; i++) {*

*var newGameMove = newGameMoves[i];*

*game.ugly\_move(newGameMove);*

*var boardValue = -evaluateBoard(game.board())*

*game.undo();*

*if (boardValue > bestValue) {*

*bestValue = boardValue;*

*bestMove = newGameMove*

*}*

*}*

*return bestMove;*

*};*

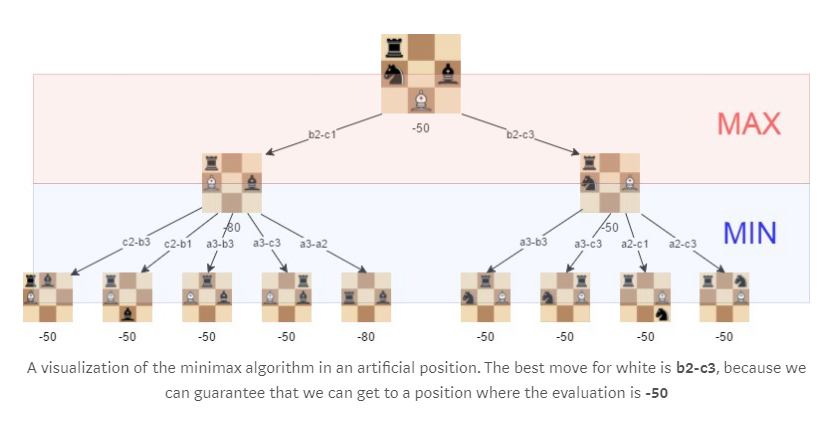
1. ***Search tree using Minimax***

We’re going to create a search tree from which the algorithm can chose the best move. This is done by using the Minimax algorithm.

In this algorithm, the recursive tree of all possible moves is explored to a given depth, and the position is evaluated at the ending “leaves” of the tree.

After that, we return either the smallest or the largest value of the child to the parent node, depending on whether it’s a white or black to move.

In a Chess Game, White is the maximizer and Black is the minimizer. With minimax in place, the algorithm is starting to understand some basic tactics of chess:



***Algorithm:***

*var minimax = function (depth, game, isMaximisingPlayer) {*

*if (depth === 0) {*

*return -evaluateBoard(game.board());*

*}*

*var newGameMoves = game.ugly\_moves();*

*if (isMaximisingPlayer) {*

*var bestMove = -9999;*

*for (var i = 0; i < newGameMoves.length; i++) {*

*game.ugly\_move(newGameMoves[i]);*

*bestMove = Math.max(bestMove, minimax(depth - 1, game, !isMaximisingPlayer));*

*game.undo();*

*}*

*return bestMove;*

*} else {*

*var bestMove = 9999;*

*for (var i = 0; i < newGameMoves.length; i++) {*

*game.ugly\_move(newGameMoves[i]);*

*bestMove = Math.min(bestMove, minimax(depth - 1, game, !isMaximisingPlayer));*

*game.undo();*

*}*

*return bestMove;*

*}*

*};*

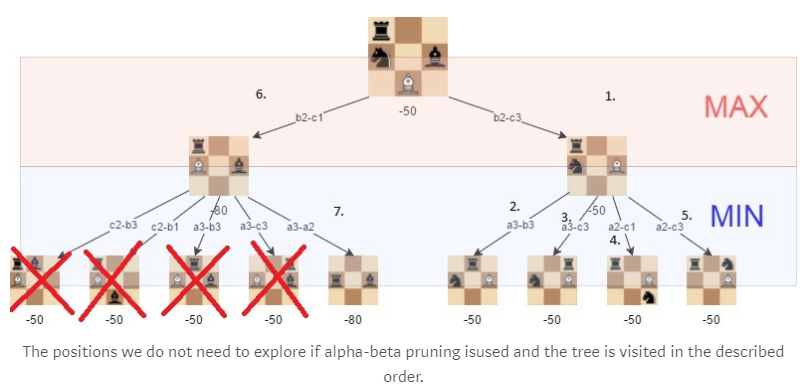
1. ***Alpha-beta pruning***

Alpha-beta pruning is an optimization method to the minimax algorithm that allows us to disregard some branches in the search tree. This helps us evaluate the minimax search tree much deeper, while using the same resources.

The alpha-beta pruning is based on the situation where we can stop evaluating a part of the search tree if we find a move that leads to a worse situation than a previously discovered move.

The alpha-beta pruning does not influence the outcome of the minimax algorithm — it only makes it faster.

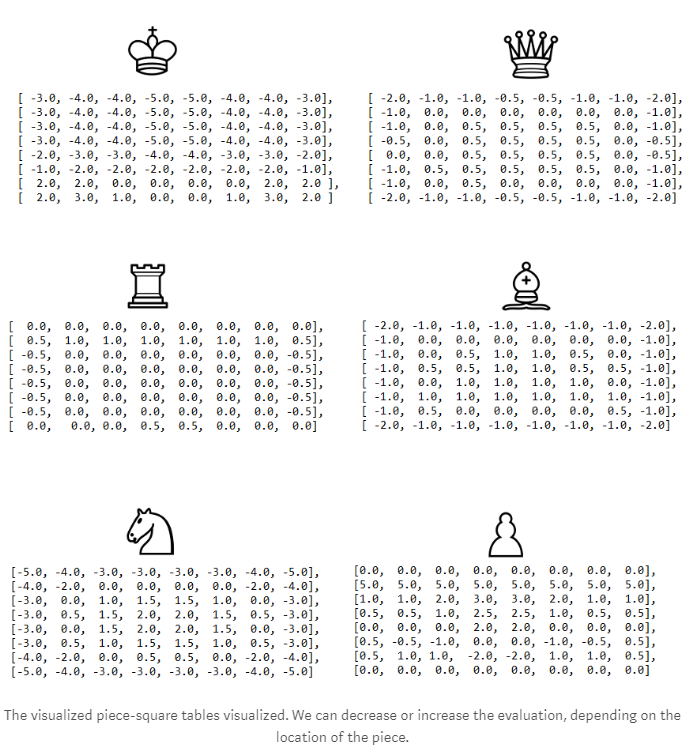
The alpha-beta algorithm also is more efficient if we happen to visit **first** those paths that lead to good moves.



1. ***Improved evaluation function:***

The initial evaluation function is quite naive as we only count the material that is found on the board. To improve this, we add to the evaluation a factor that takes in account the position of the pieces. For example, a knight on the center of the board is better (because it has more options and is thus more active) than a knight on the edge of the board.

We’ll use a slightly adjusted version of piece-square tables that are shown below.



***Results:***

The strength of even a simple chess-playing algorithm is that it doesn’t make stupid mistakes.

With the methods used here, we’ve been able to program a chess-playing-algorithm that can play basic chess. The “AI-part” (move-generation excluded) of the final algorithm is just 200 lines of code, meaning the basic concepts are quite simple to implement.

We are able to evaluate upto a depth of 5 which makes it really a tough opponent to beat.

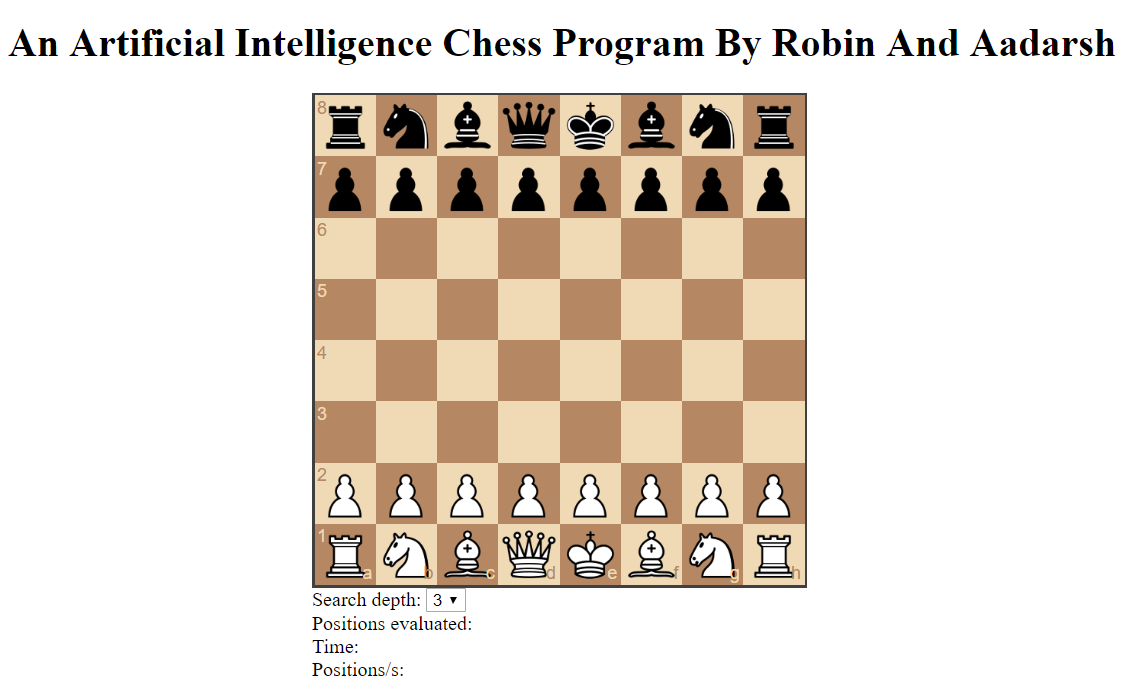
Some further improvements we could make to the algorithm would be for instance:

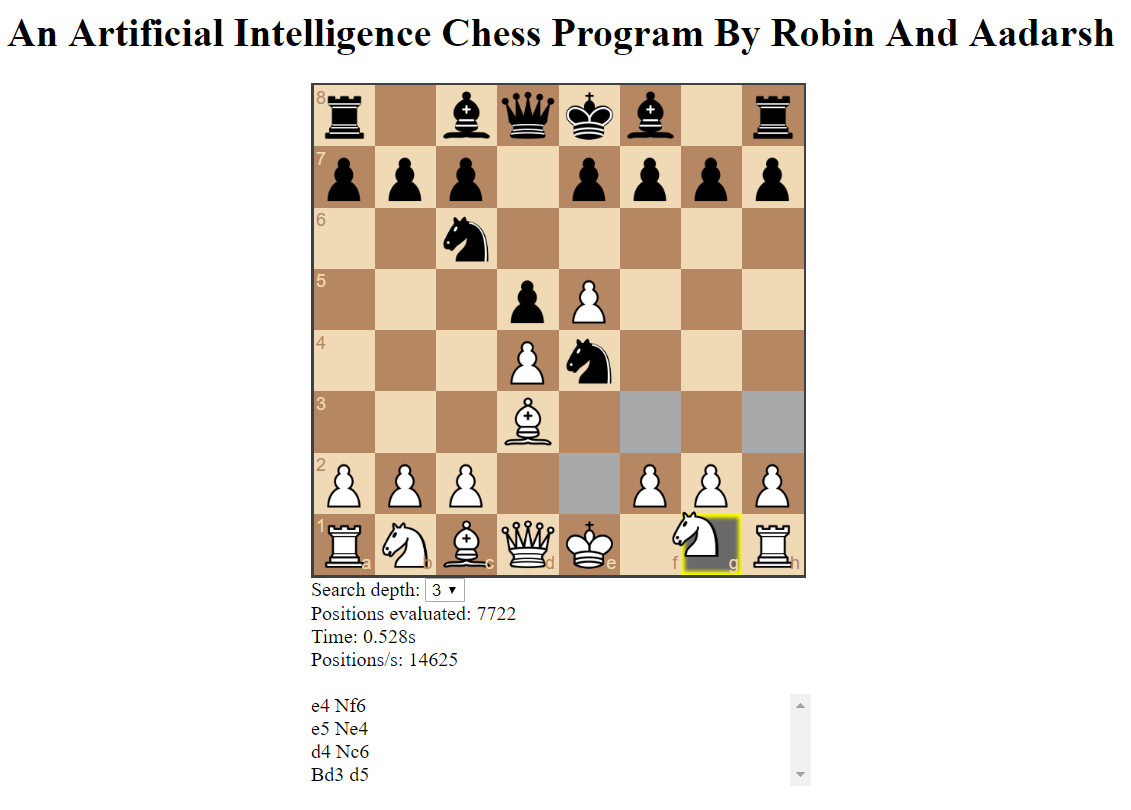
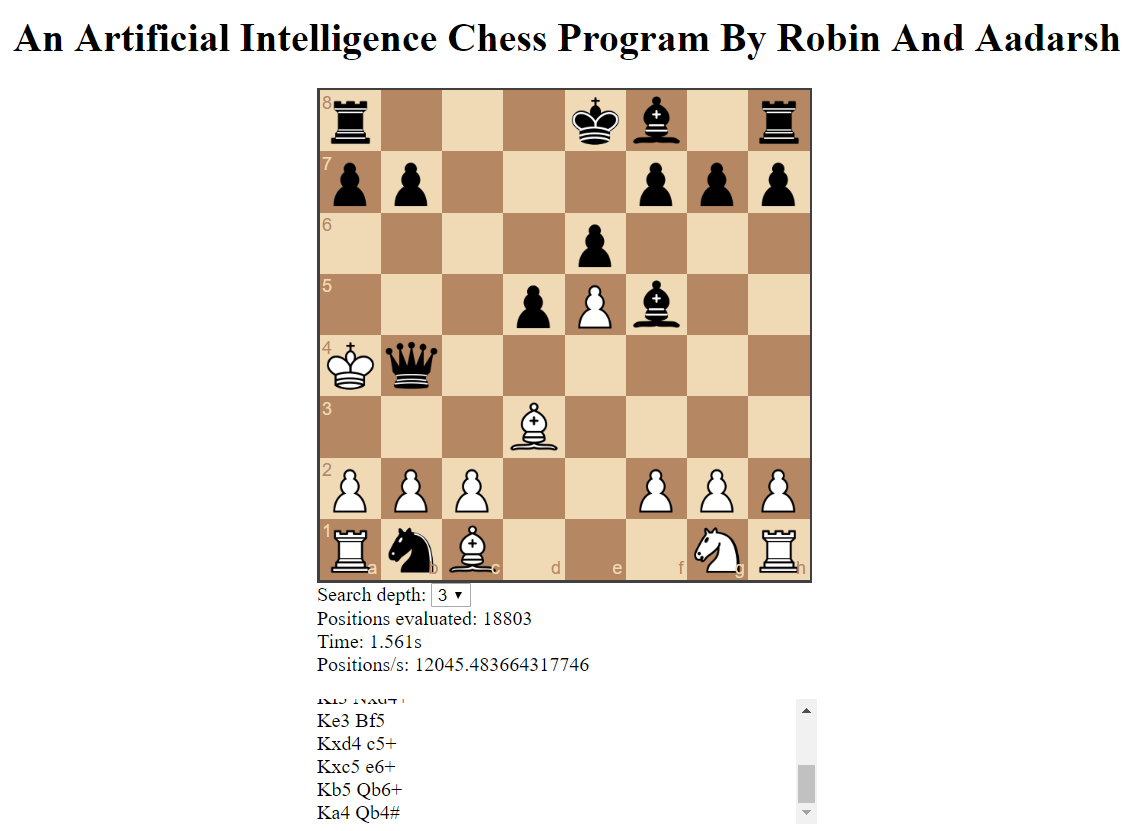
* move-ordering
* faster move generation
* and end-game specific evaluation

One of the major bottlenecks in the current version is determining legal moves at each branch, and scoring the board. This must be done for each board position encountered, which of course grows exponentially with the ply search depth. The rules of chess are relatively complex, particularly with special moves such as castles and en passant captures, and the concept of checking a king. Scoring a board is even more difficult, as not only valid moves for each piece must be determined, but also what pieces are protected and threatened in each position. It is possible to speed up all these calculations by creating a different representation of the chess board. Years of research have yielded viable so-called “bit boards,” which keep track of different aspects and properties of the board in data structures that allow for very fast computations. Re-implementing the program with such bit boards would certainly improve it.

Knowledge databases of opening moves and end-game positions could be added. Chess players over the years have discovered that certain move combinations at the beginning of games will develop into good positions mid-game. Since these are well-known, hard-coding appropriate responses would greatly improve the program’s performance in the initial phase of the game – which would then extend into an improved mid-game. Analogously, at the end of the game, when there are only a few pieces left on the board, it is possible to explicitly reach the goal state via local searches, thus determining optimal moves for many possible positions. Putting these responses into a database would allow the program to perform flawlessly in the end-game.

***Screenshots:***



#### **References:**

**Official rules of Chess**  
<http://www.uschess.org/beginners/letsplay.html>  
  
**Tree searches in chess**  
[http://www.xs4all.nl/~verhelst/chess/search.html](http://www.xs4all.nl/%7Everhelst/chess/search.html)  
  
**Alpha-beta pruning**  
[http://pages.cpsc.ucalgary.ca/~sueng/533/AlphaBetaPruning.html](http://pages.cpsc.ucalgary.ca/%7Esueng/533/AlphaBetaPruning.html)  
  
**Futility pruning**  
[http://supertech.lcs.mit.edu/~heinz/dt/node26.html](http://supertech.lcs.mit.edu/%7Eheinz/dt/node26.html)  
  
**Chess piece graphics**   
<http://www.chessvariants.com/graphics.dir/alfaerie/index.html>  
  
**Computer Chess Programming**   
[http://www.xs4all.nl/~verhelst/chess/programming.html](http://www.xs4all.nl/%7Everhelst/chess/programming.html)  
<http://chess.verhelst.org/>  
<http://www.gamedev.net/reference/programming/features/chess1/>  
  
  
 Computer game-playing : theory and practice / edited by M.A. Bramer. Published: Chichester, West Sussex : E. Horwood ; New York : Halsted Press, 1983.

Newborn, Monroe. Deep Blue : an artificial intelligence milestone / Momty Newborn ; foreword by Charles E. Lieserson. Published: New York : Springer, c2003.

Botvinnik, M. M. (Mikhail Moiseevich), Computers in chess : solving inexact search problems / M.M. Botvinnik ; with contributions by A.I. Reznitsky ... [et al.] ; translated by Arthur A. Brown. Published: New York : Springer-Verlag, c1984.

Atkinson, George W. Chess and machine intuition / George W. Atkinson. Published: Norwood, N.J. : Ablex Pub., 1993.