# NEA FabioChess

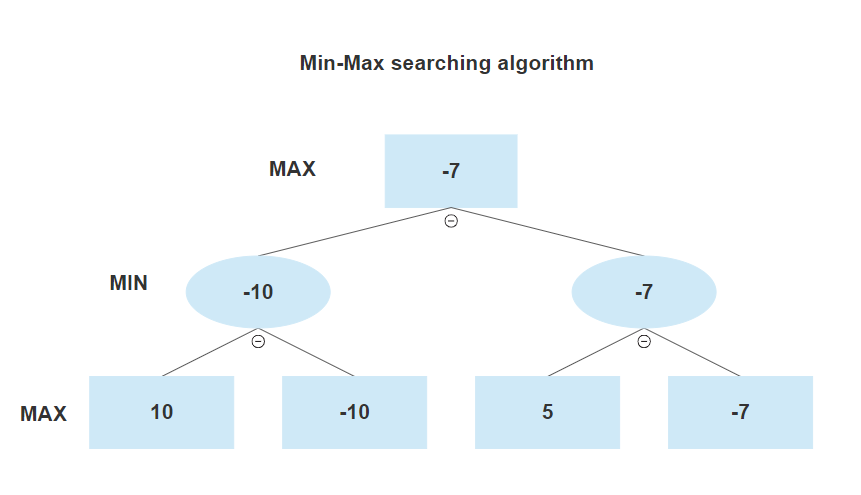
## Analysis

I will be making a website where you can play chess against a fairly strong engine. I researched existing projects such as VueChess, PyChess, SunFish and Python-Chess for this project. I also researched a lot of the technical things about chess engine programming on chessprogramming.org

I am making this program for regular chess players, so they can improve their skills against an engine which is suited for their level. I’m going to be working with the chess club in our school for their feedback throughout this project.

### Objectives:

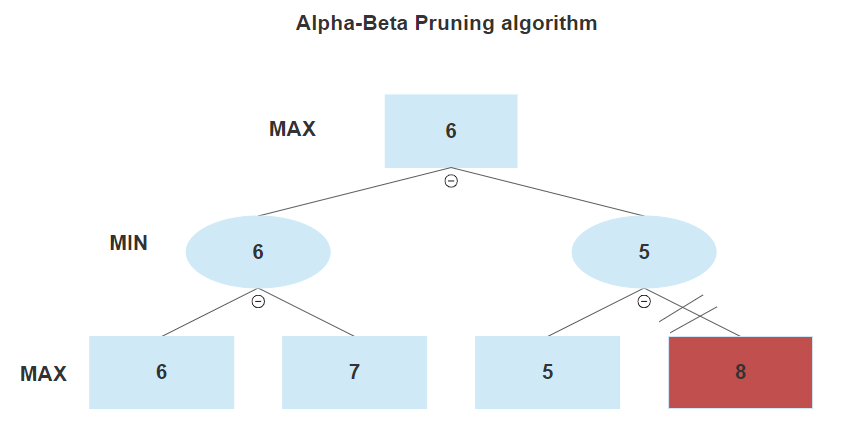
1. All the chess pieces should behave as the rules are set
2. The user should be able to use drag and drop to move the chess pieces
3. It should be able to limit the player to only be able to move in legal positions
4. Player should be able to choose from a variety of puzzles



MAX – This means it is the players turn and you choose the highest value move

MIN – This means it is the opponents turn and they will choose the move which has the lowest value.

I will also be using the alpha-beta pruning algorithm which is an improvement to the Min-Max algorithm.



// - This means the node is not searched because there is already a higher value choice

This cuts down the amount of nodes which needs to be searched by a lot.

## Design

### Algorithms

* Min-Max Searching algorithm:
* Alpha-Beta Pruning algorithm:
* Perft-testing:

### Data structures

* Board representation:
  + Algebraic notation:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | A1 | B1 | C1 | D1 | E1 | F1 | G1 | H1 |
| 2 | A2 | B2 | C2 | D2 | E2 | F2 | G2 | H2 |
| 3 | A3 | B3 | C3 | D3 | E3 | F3 | G3 | H3 |
| 4 | A4 | B4 | C4 | D4 | E4 | F4 | G4 | H4 |
| 5 | A5 | B5 | C5 | D5 | E5 | F5 | G5 | H5 |
| 6 | A6 | B6 | C6 | D6 | E6 | F6 | G6 | H6 |
| 7 | A7 | B7 | C7 | D7 | E7 | F7 | G7 | H7 |
| 8 | A8 | B8 | C8 | D8 | E8 | F8 | G8 | H8 |
|  | a | b | c | d | e | f | g | h |

Each rank is assigned a digit 1 to 8 and each file is assigned a letter a to h

* + 64 Square board representation:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 2 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 3 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| 4 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 5 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |
| 6 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| 7 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 |
| 8 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
|  | a | b | c | d | e | f | g | h |

Each file in each rank is represented by a digit ranging from 0 to 63

* + 120 Square board representation:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
| 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |
| 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 |
| 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 |
| 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 |
| 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 |
| 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 |
| 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 |
| 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 |
| 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 |

Each square is represented with a digit from 0 to 119, the actual chess board coordinates are located in the middle of the array. I will be using this king of representation to make it easier to detect if a move is possible or not so when the engine is searching for moves for a knight on the edge of the board than it can easily tell if the move is on the board of off board.

* + Representing pieces for the engine:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 4 | 2 | 3 | 5 | 6 | 3 | 2 | 4 |
| 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |
| 8 | 10 | 8 | 9 | 11 | 12 | 9 | 8 | 10 |
|  | a | b | c | d | e | f | g | h |

When a square has no piece on it, it is represented with a 0, white pawns are represented with a 1, white knights are represented with a 2, white bishops are represented with a 3, white rooks are represented with a 4, white queens are represented with a 5, the white king is represented with a 6 and the black pieces are represented with digits 7 to 12.

* + FEN string:

Forsyth-Edwards Notation is a standard notation for describing a particular board position of a chess game. The main use of FEN is to be able to restart a game from any particular position, I will use FEN to store the initial position of the board as well as using it to represent the puzzles. A FEN string is made up of 6 parts, these parts are separated by spaces.

1. The 1st part is the piece positions on the board, it describes each rank starting from rank 8 to rank 1. Pieces are represented by “P” for pawn “R” for rook “N” for knight “B” for bishop “Q” for queen and “K” for king. If a piece is white than the letter representing the piece is a capital and if the piece is black than it’s a lowercase letter. Empty squares are represented using digits 1 through 8 according to how many empty squares there are in a row and “/” divides each rank.
2. The 2nd part is which side has to move next, “w” means white moves next and “b” means black moves next
3. The 3rd part represents the castling rights. If neither side can castle than the representation will be “-“. Otherwise if white can castle kingside than this is represented by a “K” and if white can castle queen side than it is represented as “Q”, this is the same for black just with lowercase letters.
4. The 4th part represents the En passant target square in algebraic notation. If there is no en passant target square than this is represented with “-“. Every time a pawn makes a two-square move than the position behind the pawn is recorded.
5. The 5th part represents the Halfmove clock. This is the number of halfmoves since the last capture or pawn advance. This is used to determine if a draw can be claimed under the fifty-move rule.
6. The 6th part represents the Fullmove number. The full move number is increased by one every time after black moves and starts off with a 1.

For an example this is the starting position FEN string:

rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq – 0 1

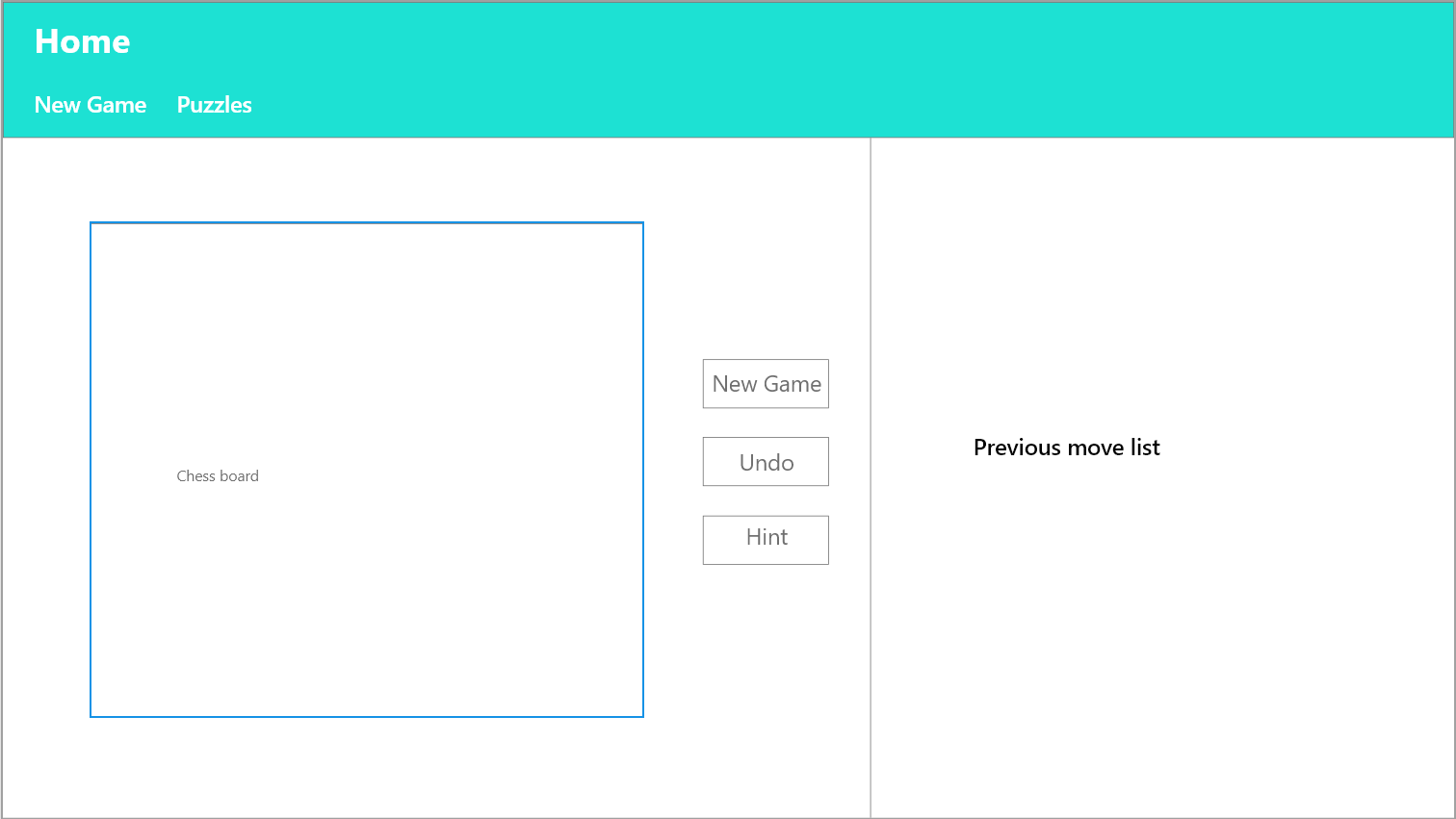
* Move number:

A single move is going to be represented by 24 bits, with these bits you can store the from square, the to square, the piece which got captured, if it was an En passant capture, if it was a pawns starting move, the promoted piece and if it was a castling move.

1. The from square is stored in the first 7 bits therefore to get the from square from the move number you can *bitwise And* the with 0x7F (0111 1111) – this extracts the first 7 bits from the move number. We need 7 bits to be able to store a digit between 0-119 which are the board coordinates.
2. The to square is stored in the 7 bits after the from square’s 7 bits. To get the to square from the move number you can shift 7 bits to the right and *bitwise And* the result with 0x7F (0111 1111), we shift 7 bits to the right so the to square bits are the first 7 bits. We need 7 bits to be able to store a digit between 0-119 which are the board coordinates.
3. The captured piece is stored in 4 bits after to square’s 7 bits. To get the captured piece from the move number you can shift 14 bits to the right and *bitwise And* the result with 0xF (1111), we shift 14 bits to the right so that the captured piece is to the first 4 bits in the move string, we need 4 bits to be able to represent all 12 king of pieces.
4. If the move is an En passant capture is stored with one bit after the 4 bits of the captured piece’s bits. To get if the move was an En passant capture or not from the move number you can *bitwise And* with 0x40000 (0100 0000 0000 0000 0000), we only need 1 bit to represent if the move was an En passant capture or not because it is either true or false.
5. If the move is a pawns first move or not is stored with one bit after the En passant capture bit. To get if the move was a pawns first move or not you can *bitwise And* with 0x80000 (1000 0000 0000 0000 0000), we only need 1 bit to represent if the move was a pawns first move because it is either true or false.
6. The promoted piece is stored with 4 bits after the pawn start bit. To get the promoted piece you can shift 20 bits to the right to get the 4 bits to the beginning of them move number, you can *bitwise And* with 0xF (1111), we need 4 bits to represent the promoted piece because we need to be able to represent all 12 pieces.

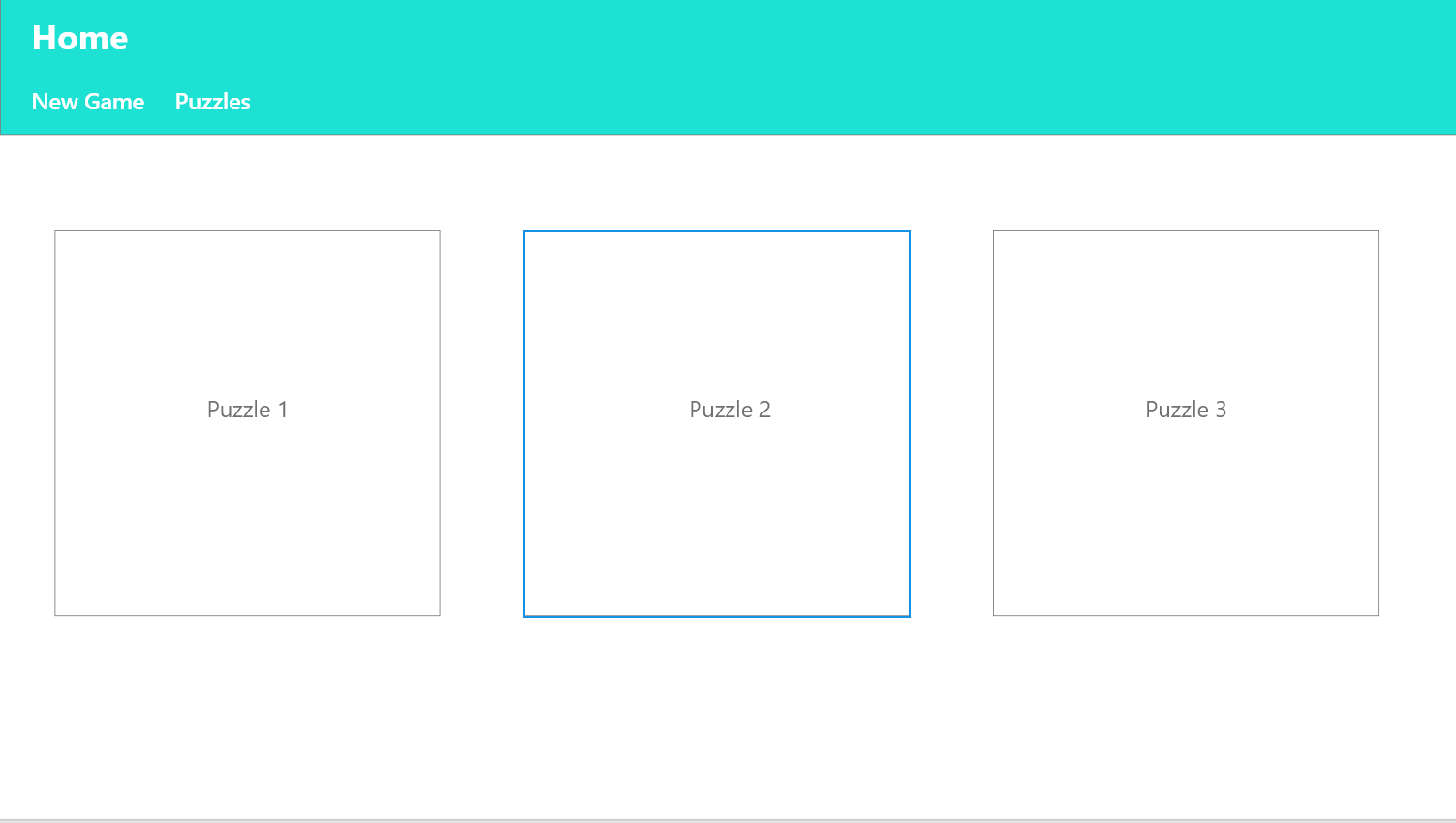
### User Interface

* Home page:



On the home page I will want to have the chess board on the left of the page with a New Game button which should start a new game, an Undo button which should undo a full move and a Hint button which should suggest to you where you should move next, to the right of the chess board. I also want to have a Previous move list on the right of the chess board and the buttons. In the navbar I want to have a New Game button as well as a Puzzles button. The New Game button should reload the page and the Puzzles button should take you to the puzzles page.

* Puzzle page:



On the puzzles page the nav bar should be the same and the user should be able to choose from different puzzles, the puzzles should be displayed as pictures of the board set up.