**Chess Engine**

**in JavaScript**

****

**­­­­**

* Ashish Papanai

**Day-1**

**Understanding the basics of chess.**

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

**Rank**

**Files**

This is a representation of an actual chess board with each Horizontal Row represents a rank.

Representation followed in this engine.

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

**Rank**

**Files**

This is a representation of a chess board according to the planned code.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 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 |

**Rank**

**Files**

Indexed representation of a chess board

Modified Board representation to prevent the piece from escaping the board

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 |

Fig 1: Indexing the modified board

If any piece reaches the yellow square, We will not allow and register that move.

Reason for adding layer of two squares on top and bottom:

Rook moves in a L shape in horizontal and vertical direction to forbade the horse from moving out of board in vertical direction we add two yellow boxes, this condition is not possible for horizontal motion of Knight, thus we have only one empty row in the left and right side to prevent.

Similarly, to prevent Rook, Bishop and Queen, we have added One layer at vertical and horizontal positions

**Day-2**

Representation used in the code:

let PIECES = {EMPTY: 0, wP: 1, wN: 2, wB: 3, wR: 4, wQ: 5, wK: 6,

              bP: 7, bN: 8, bR: 10, bQ: 11, bK: 12 };

let BRD\_SQ\_NUM = 120;

let FILES = {FILE\_A: 0, FILE\_B: 1,FILE\_C: 2,FILE\_D: 3,

    FILE\_E: 4,FILE\_F: 5,FILE\_G: 6,FILE\_H: 7, FILE\_NONE: 8};

let RANKS = {RANK\_1: 0, RANK\_2: 1,RANK\_3: 2,RANK\_4: 3,

        RANK\_5: 4,RANK\_6: 5,RANK\_7: 6,RANK\_8: 7, RANK\_NONE: 8};

var COLOURS = {WHITE: 0, BLACK: 1, BOTH:2 };

var SQUARES = {

    A1:21, B1:22, C1:23, D1:24, E1:25, F1:26, G1:27, H1:28,

    A8:91, B8:92, C8:93, D8:94, E8:95, F8:96, G8:97, H8:98,

    NO\_SQ: 99, OFFBOARD: 100

};

wP: White Pawn, wN: White Knight, wB: White Bishop, wR: White Rook, wQ: White Queen, wK: White King

bP: Black Pawn, bN: Black Knight, bB: Black Bishop, bR: Black Rook, bQ: Black Queen, bK: Black King

File representation of the board: x

Fig 2: file representation of the board

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 |
| 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 |
| 100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 100 |
| 100 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 100 |
| 100 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 100 |
| 100 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 100 |
| 100 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 100 |
| 100 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 100 |
| 100 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 100 |
| 100 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 100 |
| 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 |
| 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 |

This board represents the file according to the representation used by us in this code.

Implement conversion of files to square:

Refer fig 1 for visualization.

Initialise the board as depicted in fig 2:

function InitFilesRanksBrd(){

    let index = 0;

    let file = FILES.FILE\_A;

    var rank = RANKS.RANK\_1;

    var sq = SQUARES.A1;

    for(index = 0; index < BRD\_SQ\_NUM; ++index){

        FilesBrd[index] = SQUARES.OFFBOARD;

        RanksBrd[index] = SQUARES.OFFBOARD;

    }

    for(rank = RANKS.RANK\_1; rank <= RANKS.RANK\_8; ++rank){

        for(file = FILES.FILE\_A; file <= FILES.FILE\_H; ++file){

            sq = FR2SQ(file, rank);

            FilesBrd[sq] = file;

            RanksBrd[sq] = rank;

        }

    }

}

Castling:

let CASTLEBIT = { WKCA: 1, WQCA: 2, BKCA: 4, BQCA: 8};

White King side Castle: 0001

White Queen side Castle: 0010

Black King side Castle: 0100

Black Queen side Castle: 1000

GameBoard.castlePerm = 0; //will keep a track of castling permission.

If we want to check that castling is possible at that particular side, we’ll take bitwise AND of the corresponding values of the castling sides with the castlePerm, this will tell if that castle is allowed or not.

Snippet to keep a track of all pieces and the piece values

var PieceBig = [ BOOL.FALSE, BOOL.FALSE, BOOL.TRUE, BOOL.TRUE, BOOL.TRUE, BOOL.TRUE, BOOL.TRUE, BOOL.FALSE, BOOL.TRUE, BOOL.TRUE, BOOL.TRUE, BOOL.TRUE, BOOL.TRUE ];

var PieceMaj = [ BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.TRUE, BOOL.TRUE, BOOL.TRUE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.TRUE, BOOL.TRUE, BOOL.TRUE ];

var PieceMin = [ BOOL.FALSE, BOOL.FALSE, BOOL.TRUE, BOOL.TRUE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.TRUE, BOOL.TRUE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE ];

var PieceVal= [ 0, 100, 325, 325, 550, 1000, 50000, 100, 325, 325, 550, 1000, 50000  ]; // Value of Piece

var PieceCol = [ COLOURS.BOTH, COLOURS.WHITE, COLOURS.WHITE, COLOURS.WHITE, COLOURS.WHITE, COLOURS.WHITE, COLOURS.WHITE,

    COLOURS.BLACK, COLOURS.BLACK, COLOURS.BLACK, COLOURS.BLACK, COLOURS.BLACK, COLOURS.BLACK ];

var PiecePawn = [ BOOL.FALSE, BOOL.TRUE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.TRUE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE ];

var PieceKnight = [ BOOL.FALSE, BOOL.FALSE, BOOL.TRUE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.TRUE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE ];

var PieceKing = [ BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.TRUE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.TRUE ];

var PieceRookQueen = [ BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.TRUE, BOOL.TRUE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.TRUE, BOOL.TRUE, BOOL.FALSE ];

var PieceBishopQueen = [ BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.TRUE, BOOL.FALSE, BOOL.TRUE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.TRUE, BOOL.FALSE, BOOL.TRUE, BOOL.FALSE ];

var PieceSlides = [ BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.TRUE, BOOL.TRUE, BOOL.TRUE, BOOL.FALSE, BOOL.FALSE, BOOL.FALSE, BOOL.TRUE, BOOL.TRUE, BOOL.TRUE, BOOL.FALSE ];

Explanation of the variables:

* PieceBig: Tells if a piece is a pawn or a non-pawn piece.
* PieceMaj: Stores if a piece is a major piece, i.e. Rook and Queen
* PieceMin: Stores if a piece is a minor piece or not, i.e. Bishop and Knight
* PieceVal: stores the value of each piece

For the values of each pieces we keep pawn as a reference, Points of all pieces:

* Pawn - 100
* Knight - 325
* Bishop - 325
* Rook - 550
* Queen - 1000
* King – 50000

PieceCol: Stores the colour of Pieces.

All the elements inside the arrays are indexed according to the key value pairs mentioned in the references to variables in (page no: 5).

Note: We can have at-most 10 pieces of same type.

Algorithm for movement of pieces-

If piece in the square is of the side to move, then we’ll generate moves for that piece.

How to get index?

e.g.

wPNum or wNNum is nothing but the index of number of pieces from the list GameBoard.pceNum.

i.e. if a board has 2 white knights, then

Get location of squares containing that piece-

For(pceNum = 0; pceNum < GameBoard.pceNum[wP]; ++pceNum){

Sq = PlistArray[wP \* 10 + pceNum];

}

wP can be replaced with the pieces we want to consider and find moves of.