



SRM INSTITUTE OF SCIENCE AND
TECHNOLOGY



SCHOOL OF COMPUTING

DEPARTMENT OF DATASCIENCE AND
BUSINESS SYSTEMS

18CSC305J ARTIFICIAL INTELLIGENCE

MINI PROJECT REPORT

Title - Online Chess

NAME: AVINASH REDDY VASIPALLI

REGISTER NUMBER: RA1911027010007

MAIL ID: AV4443@SRMIST.EDU.IN

DEPARTMENT: B.TECH

SPECIALIZATION: CSE BIG DATA ANALYTICS

SEMESTER: VI

Team Members

- | | |
|-------------------------|-----------------|
| • P. Aditya | RA1911027010003 |
| • Arnav | RA1911027010040 |
| • Yuvaraj singh chauhan | RA1911027010058 |

Name

Registration Number

CONTENT PAGE

Abstract

- **Chapter 1:**
Introduction and Motivation [Purpose of the problem statement (societal benefit)]
- **Chapter 2:**
Review of Existing methods and their Limitations
- **Chapter 3:**
Proposed Method with System Architecture / Flow Diagram
- **Chapter 4:**
Modules Description
- **Chapter 5:**
Implementation requirements
- **Chapter 6:**
Output Screenshots

- Conclusion
- References

- Appendix A – Source Code
- Appendix B – GitHub Profile and Link for the Project

ABSTRACT

That one game with endless possibilities and tactics “CHESS”. We know chess is something that is beyond moves with 64 checks it has enormous possibilities. For every Pawn, rook, knight, bishop, Queen and even Single step king has endless possibilities. It is game when we play against the opponents mind directly. With basic rules for every chess piece, it looks easy but involves a lot of things like backtracking and other concept

INTRODUCTION

The main purpose of this report is to describe the web-based chess game developed by us named “IQ-Chess”. This report has the programming language and packages that are used in developing this game. The design, architecture and structuring of the code including the frontend and backend of the game.

Game rules and its functioning on AI and the encryption technique and detailed explanation about why we this this game is the best example in terms of applications for AI.

DESIGN AND ARCHITECTURE

Languages:



Front End

- JavaScript-(js)
- HTML and CSS
- Socket.io



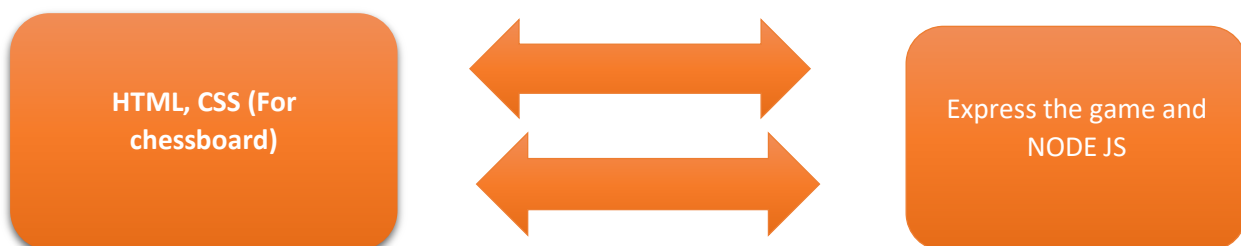
Back End

- Python
- JavaScript
- And deployment

Architecture:

Modules in the Architecture

- ✓ Realtime experience with computer on gaming logic
- ✓ Easy modulation and movement for AI
- ✓ Monitor dashboard to display the game
- ✓ Available movements present for a piece



CHAPTER 1

Introduction and Motivation

Chess is an ancient game played in different form and evolved into present modern game. With 8 pawns, 2 knights, bishops, rooks each and 1 queen and 1 king. This game mostly depends on the mind and moves the opponent use. Each players have to adopt to their opponent every step and make moves resulting their opponents king surrounded. This game mostly depends on the mind where it comes downs to predicting opponent.

With all these elements a computer intelligence is something we can rely on as it can think beyond any average human can. As we can see for every step a human make it is required to think of their next steps and his own steps this kind of thinking is easy for AI using algo like backtracking, N Queen.

Main motivation for this project is the endless possibility availability in the step we have to make. Just think if we move a pawn to any position what are the possibilities to the opponent or to the player to make his move. Existing rules like 16 step rules to 50 step rules to every situation. Even after so many possible scenarios or placement of pieces their exist some moves even the grandmasters can't think of or experienced for the first time.

This makes it the perfect example for AI. AI evolves for every step it experiences and learns from its approach this makes it ideal player a human can play against. Not every player can play with all the skills he experienced.

CHAPTER 2

Review of Existing methods and their Limitations

Windows 7 has the best online chess so far with the best user interface and overall experience. But nothing is perfect and AI evolves every time and it has its limitations some are:

- ✓ Windows 7 is no more in terms of OS.
- ✓ It has game that can not be hosted in every computer.
- ✓ It has particular set of rules i.e. if we player the same step in to different games it plays same step.

These kinds of limitations make learning stop and the steps and its counters are not noted or delt with.

Present method doesn't have the constant learning phase. Online game does provide a proper interface for the users. Apart from Windows many websites do host the game but their GUI and AI are just for the game to run but nothing more. Proper planning in terms of steps for the next move and its analysis it missing.

Our model has basic backtracking model which helps the AI to retrace its steps when any wrong step is make which might result in loss. AI when finds this backtrack to the resent step so that it can correct it step such that the lose faced is less and that can result in winning the game finally.

CHAPTER 3

Proposed Method with System Architecture / Flow Diagram

Workflow for our project is simple and easy which is mentioned below:

Phase 1

- ✓ Planning features, execution and algorithmic approaches
- ✓ Plan brainstorming
- ✓ Simple backtracking model
- ✓ BFS implementations to check the completeness of model

Phase 2

- ✓ Implementation of N Queen Problem
- ✓ GUI creation
- ✓ Chessboard and numbering making
- ✓ Functions defining for backtracking in game

Phase 3

- ✓ Testing of GUI
- ✓ Testing against Controls and test cases
- ✓ Testing of every module and as many cases as possible
- ✓ Report generation and recoding for the output

Phase 4 (Final)

- ✓ Deployment and external usage
- ✓ Commercialization and distribution
- ✓ Website deployment and GIUHUB linkage

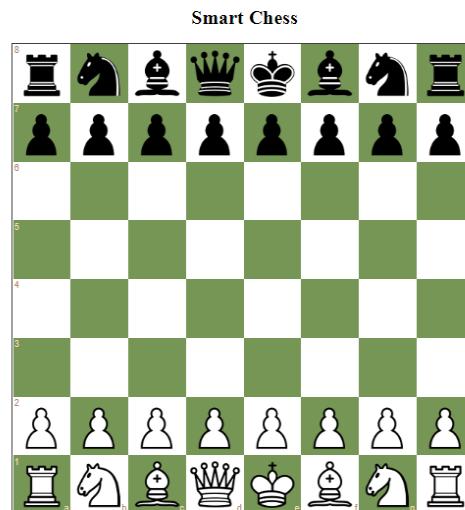
CHAPTER 4

Modules Description

Project modules and its deliverables

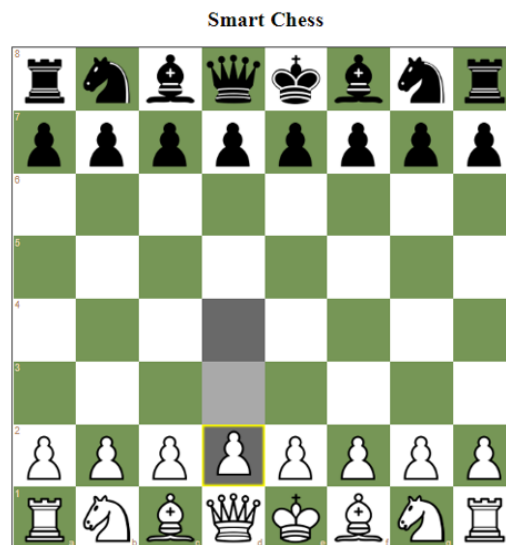
✓ User Page

Using a external website this game is histed so that it can be played in any computer anywhere in the world with internet. By default, the user gets White pieces for advantage norms. White has to make the first move according to the rules and the user page suggest that user gets white every time.



✓ User Move

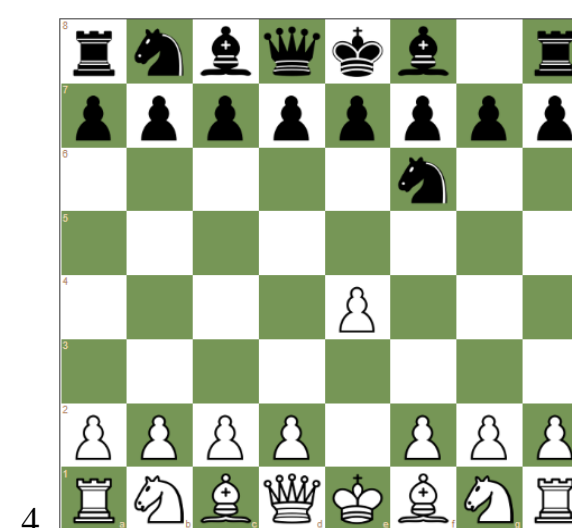
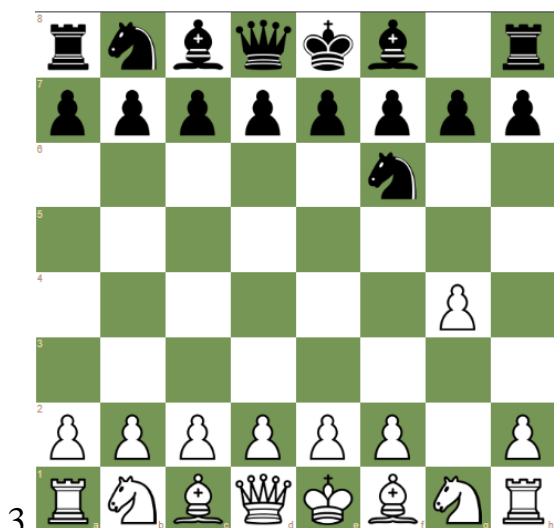
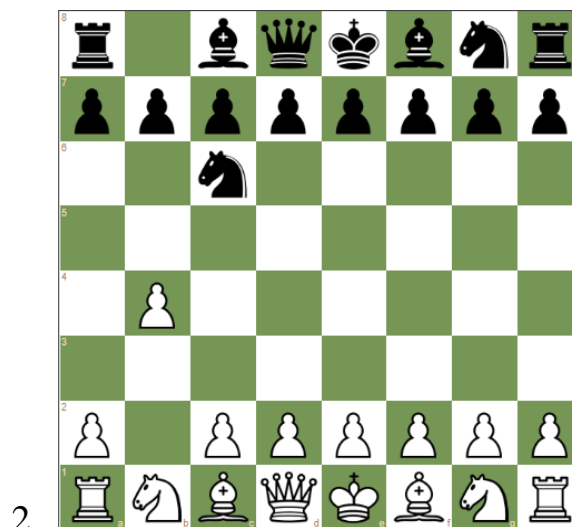
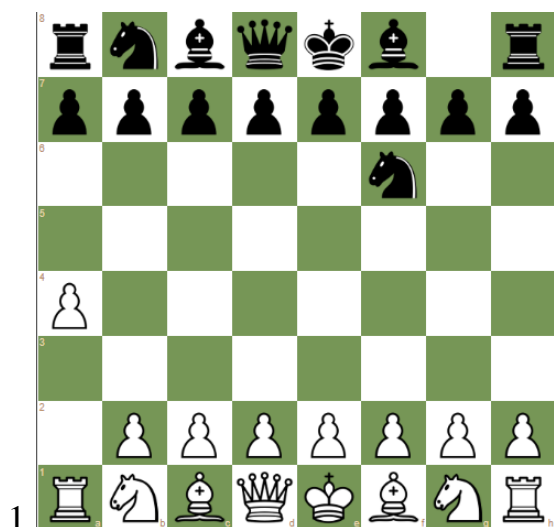
The user is expected to move first as his piece is white by default. This is true no matter how many times we play. The reason is simple system can never make a step on its own even if it tries to we have to include every possibility in a random forest which can results in some random move from the list we give but giving the user the first move helps the AI to plan its step.



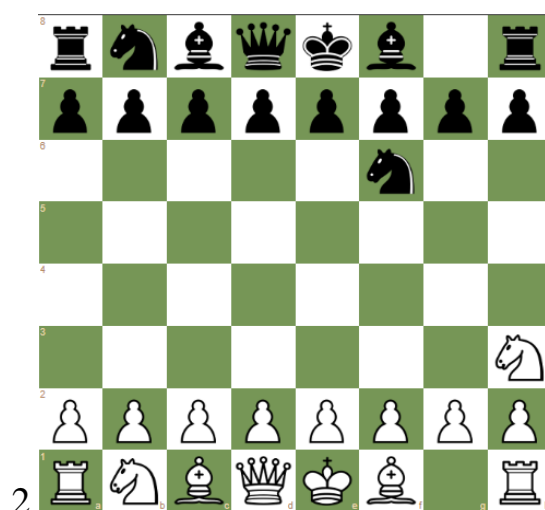
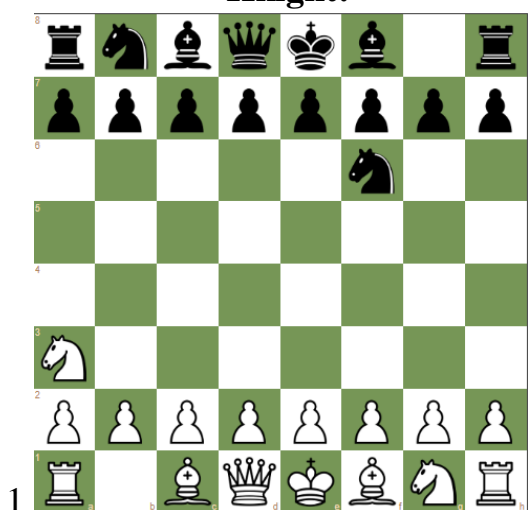
✓ System track

So, this is where the game gets interesting consider as many possibilities as we can as shown below:

• Pawns:



• Knight:



As u can see in the above images for every step the use makes the system make most suitable step in every case.

✓ Backtracking

The most important step in a game of chess BACKTRACKING. The reason is when every we feel that we might loos a piece we need to backtrack to out previous step in order to avoid the loss.

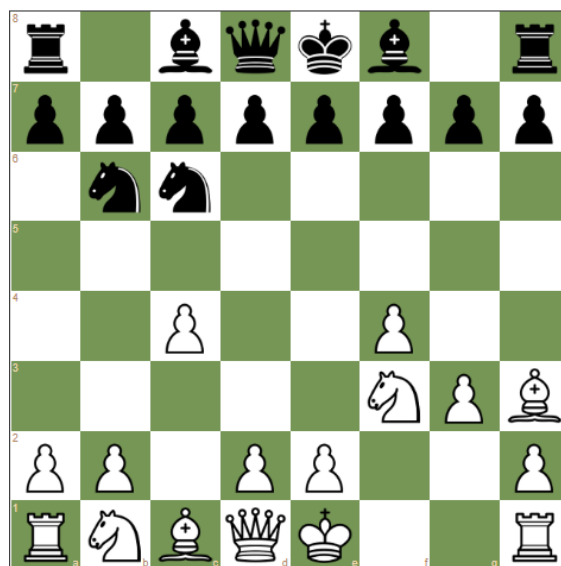
In the below step check the movement of Knights of black and pawn of white and we can find the for every move User make the AI backtracks to the safe place it can find.



(1)



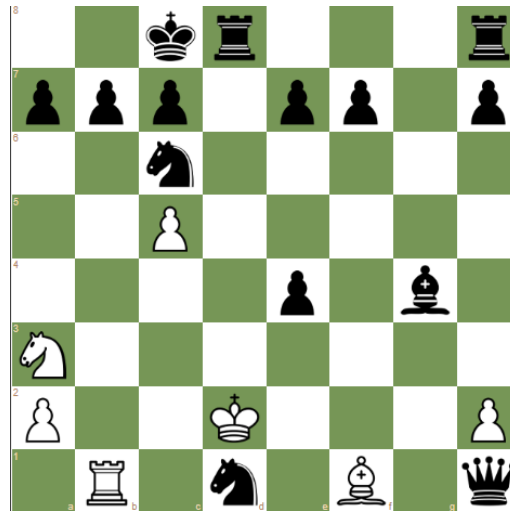
(2)



(3)

✓ Moves

Consider the no of steps a user can make this make a large data base which can be analysed for future players. Extra moves like chess castling are important at present game as they are used frequently by many players which made us implement in our game so that the user can do it and also the system when it feels like it. To have a fair and proper output we restricted the player to make single move only and can never go back once made.



✓ Update on every move

Probability of every possible move the system can make is do and best is chosen which make this process most efficient as it takes the present and past situations into consideration for this step.

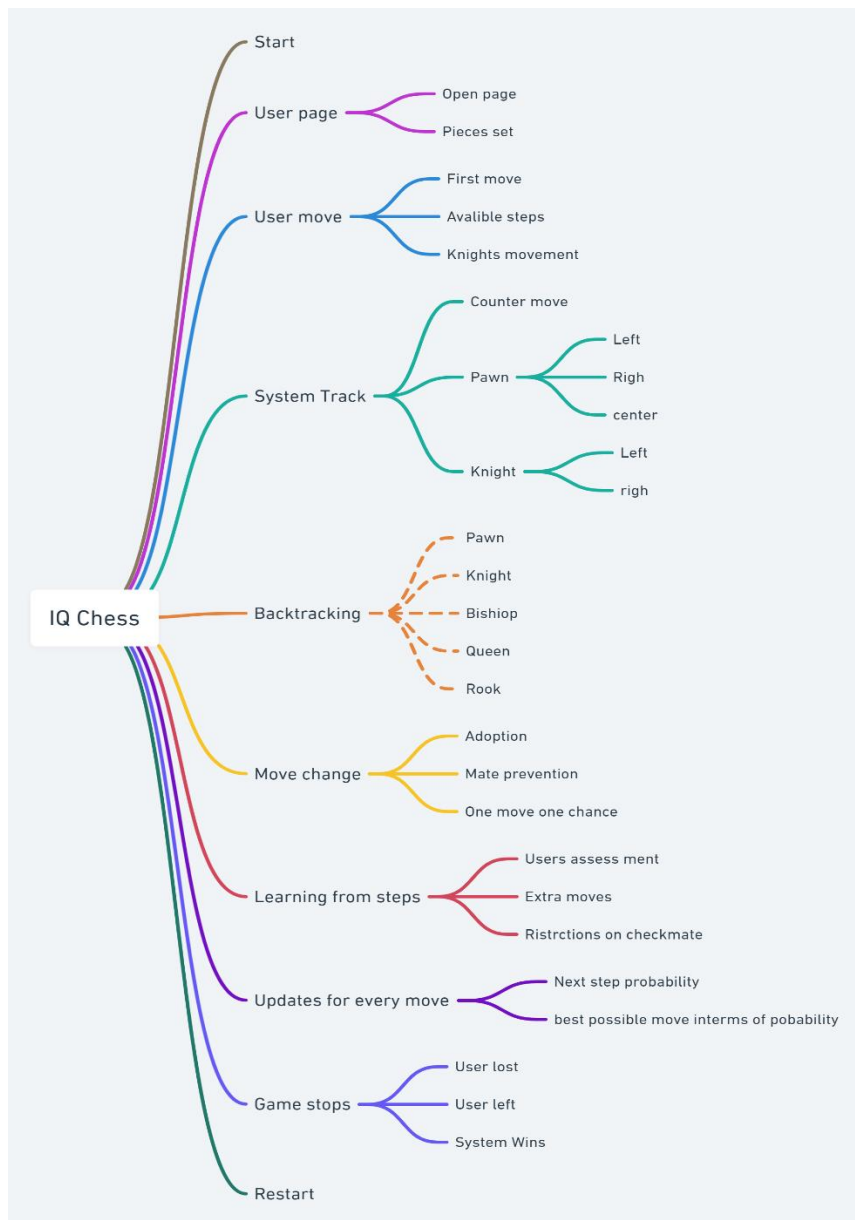


✓ Game stops

Game stop at points where User is lost, User left or the system wins. These outputs show that game is ended.



Functions in Chart

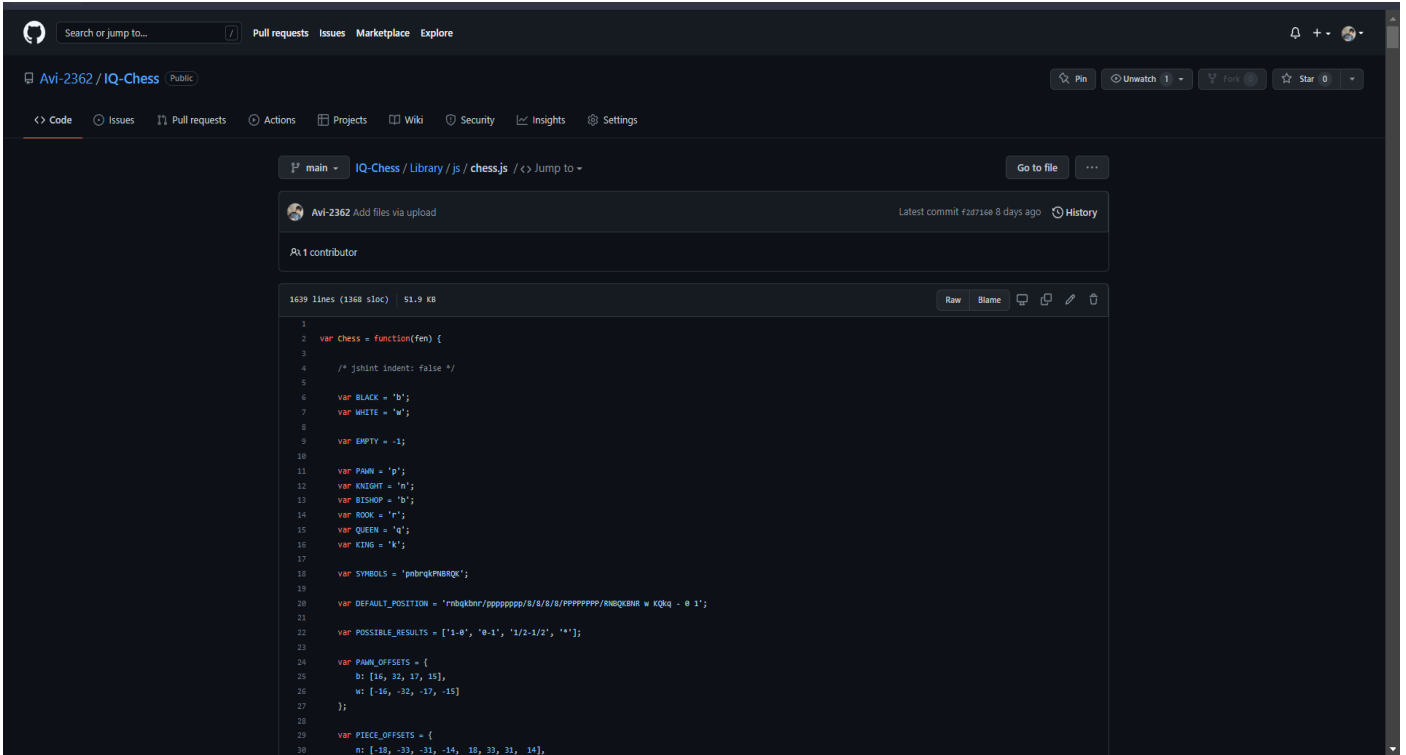


CHAPTER 5

Implementation requirements

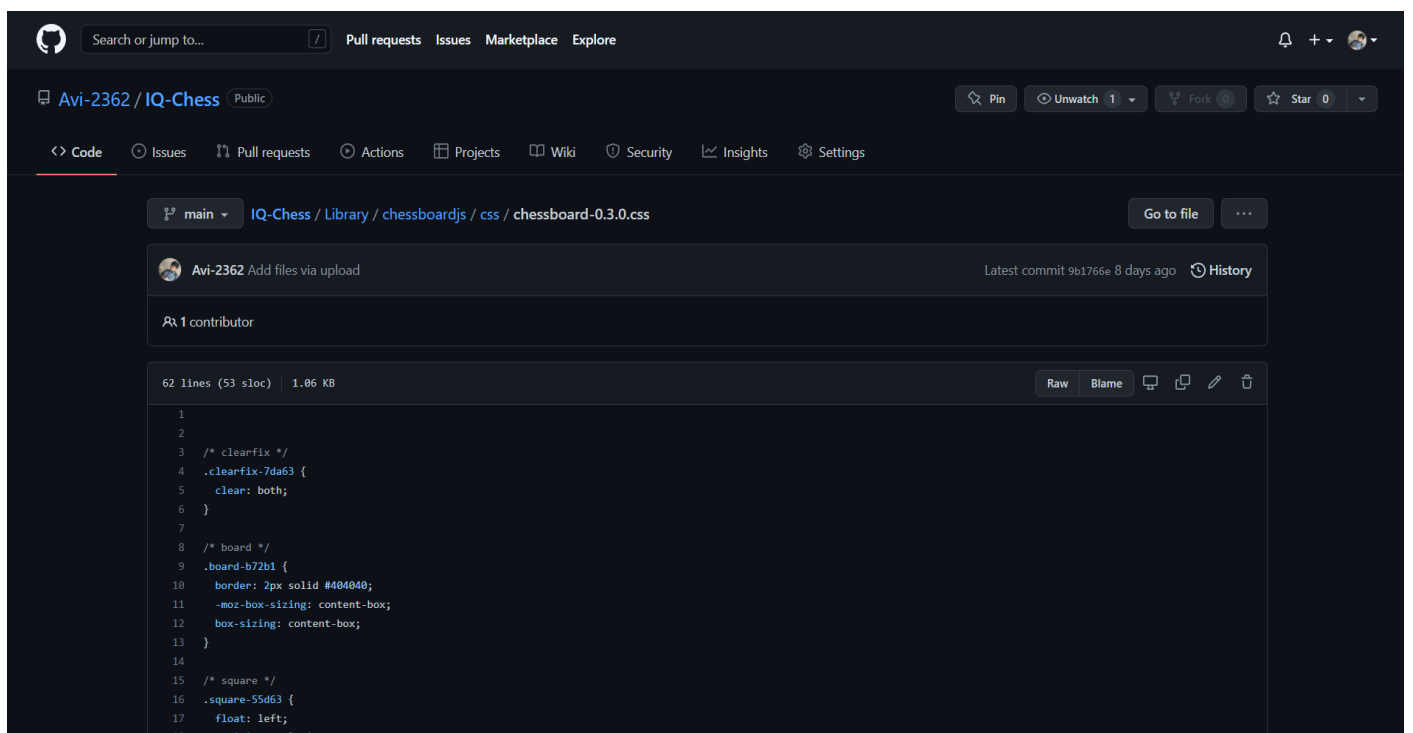
Source Code:

Js:



This screenshot shows the GitHub interface for the repository 'Avi-2362 / IQ-Chess'. The file 'chess.js' is selected, showing 1639 lines of JavaScript code. The code defines a chess engine with various constants and functions. Key elements include:



- Constants for piece colors (BLACK, WHITE), piece types (PAWN, KNIGHT, BISHOP, ROOK, QUEEN, KING), and symbols.
- A function 'chess' that takes a fen string as input.
- Arrays for 'POSSIBLE_RESULTS' and 'PAWN_OFFSETS'.
- A function 'PIECE_OFFSETS' that returns an array of offsets for each piece type.



This screenshot shows the GitHub interface for the repository 'Avi-2362 / IQ-Chess'. The file 'chessboard-0.3.0.css' is selected, showing 62 lines of CSS code. The code defines styles for the chessboard and pieces. Key elements include:

- A class '.clearfix' for clearing floats.
- A class '.board' for the chessboard, with a border and box-sizing.
- A class '.square' for individual squares, with float and width.


HTML

 Avi-2362 Add files via upload Latest commit 0da8173 8 days ago  History

1 contributor

20 lines (19 sloc) | 571 Bytes

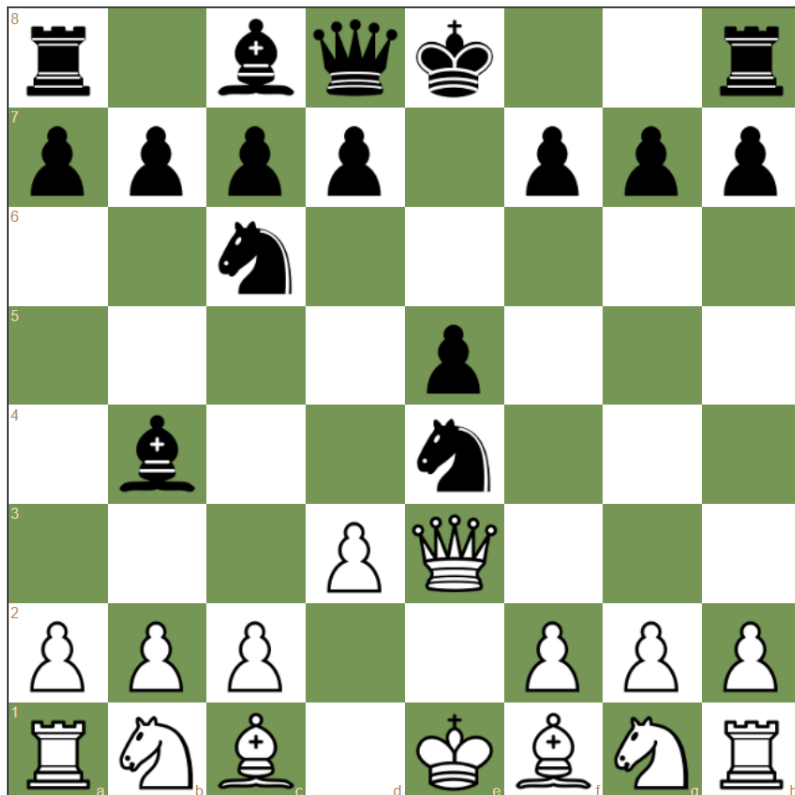
RawBlame



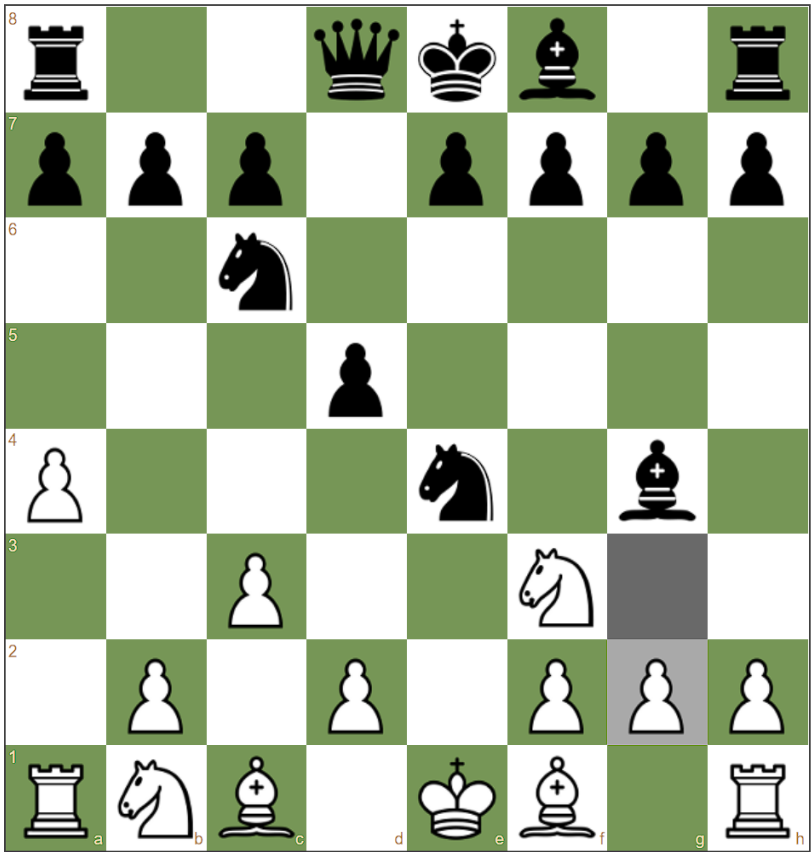
```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <link rel="stylesheet" href="lib/chessboardjs/css/chessboard-0.3.0.css">
6   <link rel="stylesheet" href="style.css">
7 </head>
8 <center><h1>Smart Chess</h1></center>
9 <body>
10 <div id="board" class="board"></div>
11 <div id="move-history" class="move-history"></div>
12 <script src="lib/jquery/jquery-3.2.1.min.js"></script>
13 <script src="lib/chessboardjs/js/chess.js"></script>
14 <script src="lib/chessboardjs/js/chessboard-0.3.0.js"></script>
15 <script src="script.js"></script>
16 <script>
17
18 </script>
19 </body>
20 </html>
```

CHAPTER 6 Output Screenshot

Smart Chess



Smart Chess



Smart Chess



SCHEDULE OF THE PROJECT

- 10th Feb 2022 - Start of the project
- 11th Feb to 19th Feb -N Queens and Solution by Backtracking
- 20th Feb to 26th Feb - Automation for optimized backtracking
- 27th Feb to 04th Mar -GUI Implementation and Deployment

CONCLUSION

Chess is game of possibilities no matter how hard we try there is always a point where the board has new positions this is what fantasize the most. In the end we made a game that is fully functional.

The use of languages like Python or C/C++ is limited because the directory available online made it easy and efficient in making this game.

REFERENCE

- ✓ Online GitHub and stack flow for the doubts part
 - <https://chess0.herokuapp.com/>
 - <https://github.com/Aveek-Saha/Online-Chess>
 - <https://stackoverflow.com/questions/7416732/online-chess-by-vb-net>
 - <https://www.chess.com/forum/view/help-support/stack-overflow>

- ✓ YouTube references on games and website deployment
 - <https://www.youtube.com/watch?v=HvjsMrR51lg>
 - <https://www.youtube.com/watch?v=QwUZxCBtFLw>

With other references from AI textbooks and research papers this project was made.

- ✓ Classmate reference mostly from Mainak and his Github
 - <https://github.com/MainakRepositor>
- ✓ Geeks for geeks on N Queens and Backtracking
 - <https://www.geeksforgeeks.org/n-queen-problem-backtracking-3/>
 - <https://www.geeksforgeeks.org/design-a-chess-game/>

APPENDIX A – SOURCE CODE

```
var board,
    game = new Chess();

/*The "AI" part starts here */

var minimaxRoot =function(depth, game, isMaximisingPlayer) {

    var newGameMoves = game.ugly_moves();
    var bestMove = -9999;
    var bestMoveFound;

    for(var i = 0; i < newGameMoves.length; i++) {
        var newGameMove = newGameMoves[i]
        game.ugly_move(newGameMove);
        var value = minimax(depth - 1, game, -10000, 10000, !isMaximisingPlayer);
        game.undo();
        if(value >= bestMove) {
            bestMove = value;
            bestMoveFound = newGameMove;
        }
    }
    return bestMoveFound;
};

var minimax = function (depth, game, alpha, beta, isMaximisingPlayer) {
    positionCount++;
    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, alpha, beta,
!isMaximisingPlayer));
        game.undo();
        alpha = Math.max(alpha, bestMove);
        if (beta <= alpha) {
            return bestMove;
        }
    }
    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, alpha, beta,
!isMaximisingPlayer));
        game.undo();
        beta = Math.min(beta, bestMove);
        if (beta <= alpha) {
            return bestMove;
        }
    }
    return bestMove;
}
};

```

```

var evaluateBoard = function (board) {
    var totalEvaluation = 0;
    for (var i = 0; i < 8; i++) {
        for (var j = 0; j < 8; j++) {
            totalEvaluation = totalEvaluation + getPieceValue(board[i][j], i ,j);
        }
    }
    return totalEvaluation;
};

```

```

var reverseArray = function(array) {
    return array.slice().reverse();
};

```

```

var pawnEvalWhite =

```

```
[
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    [5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0],
    [1.0, 1.0, 2.0, 3.0, 3.0, 2.0, 1.0, 1.0],
    [0.5, 0.5, 1.0, 2.5, 2.5, 1.0, 0.5, 0.5],
    [0.0, 0.0, 0.0, 2.0, 2.0, 0.0, 0.0, 0.0],
    [0.5, -0.5, -1.0, 0.0, 0.0, -1.0, -0.5, 0.5],
    [0.5, 1.0, 1.0, -2.0, -2.0, 1.0, 1.0, 0.5],
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
];
```

```
var pawnEvalBlack = reverseArray(pawnEvalWhite);
```

```
var knightEval =
[
    [-5.0, -4.0, -3.0, -3.0, -3.0, -3.0, -4.0, -5.0],
    [-4.0, -2.0, 0.0, 0.0, 0.0, 0.0, -2.0, -4.0],
    [-3.0, 0.0, 1.0, 1.5, 1.5, 1.0, 0.0, -3.0],
    [-3.0, 0.5, 1.5, 2.0, 2.0, 1.5, 0.5, -3.0],
    [-3.0, 0.0, 1.5, 2.0, 2.0, 1.5, 0.0, -3.0],
    [-3.0, 0.5, 1.0, 1.5, 1.5, 1.0, 0.5, -3.0],
    [-4.0, -2.0, 0.0, 0.5, 0.5, 0.0, -2.0, -4.0],
    [-5.0, -4.0, -3.0, -3.0, -3.0, -3.0, -4.0, -5.0]
];
```

```
var bishopEvalWhite = [
    [-2.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -2.0],
    [-1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -1.0],
    [-1.0, 0.0, 0.5, 1.0, 1.0, 0.5, 0.0, -1.0],
    [-1.0, 0.5, 0.5, 1.0, 1.0, 0.5, 0.5, -1.0],
    [-1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, -1.0],
    [-1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, -1.0],
    [-1.0, 0.5, 0.0, 0.0, 0.0, 0.0, 0.5, -1.0],
    [-2.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -2.0]
];
```

```
var bishopEvalBlack = reverseArray(bishopEvalWhite);
```

```
var rookEvalWhite = [
    [ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
```

```

    [ 0.5, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.5],
    [-0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.5],
    [-0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.5],
    [-0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.5],
    [-0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.5],
    [-0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.5],
    [ 0.0, 0.0, 0.0, 0.5, 0.5, 0.0, 0.0, 0.0]
];

var rookEvalBlack = reverseArray(rookEvalWhite);

var evalQueen = [
    [-2.0, -1.0, -1.0, -0.5, -0.5, -1.0, -1.0, -2.0],
    [-1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -1.0],
    [-1.0, 0.0, 0.5, 0.5, 0.5, 0.5, 0.0, -1.0],
    [-0.5, 0.0, 0.5, 0.5, 0.5, 0.5, 0.0, -0.5],
    [ 0.0, 0.0, 0.5, 0.5, 0.5, 0.5, 0.0, -0.5],
    [-1.0, 0.5, 0.5, 0.5, 0.5, 0.5, 0.0, -1.0],
    [-1.0, 0.0, 0.5, 0.0, 0.0, 0.0, 0.0, -1.0],
    [-2.0, -1.0, -1.0, -0.5, -0.5, -1.0, -1.0, -2.0]
];

var kingEvalWhite = [

    [-3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0],
    [-3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0],
    [-3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0],
    [-3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0],
    [-2.0, -3.0, -3.0, -4.0, -4.0, -3.0, -3.0, -2.0],
    [-1.0, -2.0, -2.0, -2.0, -2.0, -2.0, -2.0, -1.0],
    [ 2.0, 2.0, 0.0, 0.0, 0.0, 0.0, 2.0, 2.0 ],
    [ 2.0, 3.0, 1.0, 0.0, 0.0, 1.0, 3.0, 2.0 ]
];

var kingEvalBlack = reverseArray(kingEvalWhite);

var getPieceValue = function (piece, x, y) {

```

```

if (piece === null) {
    return 0;
}
var getAbsoluteValue = function (piece, isWhite, x ,y) {
    if (piece.type === 'p') {
        return 10 + ( isWhite ? pawnEvalWhite[y][x] : pawnEvalBlack[y][x] );
    } else if (piece.type === 'r') {
        return 50 + ( isWhite ? rookEvalWhite[y][x] : rookEvalBlack[y][x] );
    } else if (piece.type === 'n') {
        return 30 + knightEval[y][x];
    } else if (piece.type === 'b') {
        return 30 + ( isWhite ? bishopEvalWhite[y][x] : bishopEvalBlack[y][x] );
    } else if (piece.type === 'q') {
        return 90 + evalQueen[y][x];
    } else if (piece.type === 'k') {
        return 900 + ( isWhite ? kingEvalWhite[y][x] : kingEvalBlack[y][x] );
    }
    throw "Unknown piece type: " + piece.type;
};

var absoluteValue = getAbsoluteValue(piece, piece.color === 'w', x ,y);
return piece.color === 'w' ? absoluteValue : -absoluteValue;
};

/* board visualization and games state handling */

var onDragStart = function (source, piece, position, orientation) {
    if (game.in_checkmate() === true || game.in_draw() === true ||
        piece.search(/^b/) !== -1) {
        return false;
    }
};

var makeBestMove = function () {
    var bestMove = getBestMove(game);
    game.ugly_move(bestMove);
    board.position(game.fen());
    renderMoveHistory(game.history());
    if (game.game_over()) {

```

```

    alert('Game over');
  }
};

var positionCount;
var getBestMove = function (game) {
  if (game.game_over()) {
    alert('Game over');
  }

  positionCount = 0;
  // var depth = parseInt($('#search-depth').find(':selected').text());
  var depth = 3;

  var d = new Date().getTime();
  var bestMove = minimaxRoot(depth, game, true);
  var d2 = new Date().getTime();
  var moveTime = (d2 - d);
  var positionsPerS = ( positionCount * 1000 / moveTime);

  // $('#position-count').text(positionCount);
  // $('#time').text(moveTime/1000 + 's');
  // $('#positions-per-s').text(positionsPerS);
  return bestMove;
};

var renderMoveHistory = function (moves) {
  var historyElement = $('#move-history').empty();
  historyElement.empty();
  // for (var i = 0; i < moves.length; i = i + 2) {
  //   historyElement.append('<span>' + moves[i] + ' ' + ( moves[i + 1] ? moves[i + 1] :
' ') + '</span><br>')
  // }
  historyElement.scrollTop(historyElement[0].scrollHeight);

};

var onDrop = function (source, target) {

```

```

var move = game.move({
    from: source,
    to: target,
    promotion: 'q'
});

removeGreySquares();
if (move === null) {
    return 'snapback';
}

renderMoveHistory(game.history());
window.setTimeout(makeBestMove, 250);
};

var onSnapEnd = function () {
    board.position(game.fen());
};

var onMouseoverSquare = function(square, piece) {
    var moves = game.moves({
        square: square,
        verbose: true
    });

    if (moves.length === 0) return;

    greySquare(square);

    for (var i = 0; i < moves.length; i++) {
        greySquare(moves[i].to);
    }
};

var onMouseoutSquare = function(square, piece) {
    removeGreySquares();
};

var removeGreySquares = function() {
    $('#board .square-55d63').css('background', '');
};

```

```
};

var greySquare = function(square) {
  var squareEl = $('#board .square-' + square);

  var background = '#a9a9a9';
  if (squareEl.hasClass('black-3c85d') === true) {
    background = '#696969';
  }

  squareEl.css('background', background);
};

var cfg = {
  draggable: true,
  position: 'start',
  onDragStart: onDragStart,
  onDrop: onDrop,
  onMouseoutSquare: onMouseoutSquare,
  onMouseoverSquare: onMouseoverSquare,
  onSnapEnd: onSnapEnd
};
board = ChessBoard('board', cfg);
```

Appendix B

GitHub Profile and Link for the Project

GitHub Profile:

<https://github.com/Avi-2362>

Project Link:

<https://github.com/Avi-2362/IQ-Chess>

Website Link:

<https://smart-chess-ai.netlify.app/>