JavaScript Chess Engine.

let parentDiv = document.querySelector("#board");

let alphas = "abcdefgh".split("");

let nums = "87654321".split("");

let drag;

const pieces = {

'RW': 'static/img/pieces/Chess\_rlt45.svg',

'RB': 'static/img/pieces/black\_Chess\_rlt45.svg',

'NW': 'static/img/pieces/Chess\_nlt45.svg',

'NB': 'static/img/pieces/black\_Chess\_nlt45.svg',

'BW': 'static/img/pieces/Chess\_blt45.svg',

'BB': 'static/img/pieces/black\_Chess\_blt45.svg',

'QW': 'static/img/pieces/Chess\_qlt45.svg',

'QB': 'static/img/pieces/black\_Chess\_qlt45.svg',

'KW': 'static/img/pieces/Chess\_klt45.svg',

'KB': 'static/img/pieces/black\_Chess\_klt45.svg',

'PW': 'static/img/pieces/Chess\_plt45.svg',

'PB': 'static/img/pieces/black\_Chess\_plt45.svg'

}; explain this part of the code please this is for a chess engine in JavaScript

This part of the code defines an object named pieces, which contains the file paths for all the chess pieces images used in the game.

Each piece is represented by a two-letter code, where the first letter is the color (W for white and B for black) and the second letter is the type of the piece (R for rook, N for knight, B for bishop, Q for queen, K for king, and P for pawn).

For example, the code 'RW' represents the white rook, while 'PB' represents the black pawn.

The file paths are specified as string values, where static/img/pieces/ is the directory where the image files are stored, and the rest of the path is the name of the image file.

The image files have a naming convention based on the piece code and the angle of the piece in the image. The suffix 'lt45' or 'rt45' refers to the orientation of the piece (left or right tilted at 45 degrees).

Overall, this code sets up a mapping of the piece codes to their corresponding image files, which will be used later in the chess game interface to display the pieces on the board.

let board = [[ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ],

[ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ],

[ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ],

[ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ],

[ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ],

[ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ],

[ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ],

[ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ]]

This part of the code defines a 2D array named board with a size of 8 rows and 8 columns. Each element of the array represents a square on the chessboard and is initialized to 0.

In the context of a chess game, this array represents the initial state of the board with no pieces on it. The first row of the array corresponds to the 8th rank of the board (where the black pieces start), and the last row corresponds to the 1st rank (where the white pieces start).

Later in the game, the elements of this array will be updated to reflect the positions of the pieces on the board. A value of 0 in an array element means that the corresponding square is empty, while a value of 1 or -1 (depending on the color of the piece) means that there is a piece on that square.

For example, if the element at board[0][0] is set to 1, it means that there is a white rook on the square a8 (since the first row of the array corresponds to the 8th rank of the board, and the first column corresponds to the a-file).

class Rook{

#side;

constructor(side,piece,position,id){

this.#side = side;

this.piece = piece;

this.position = position;

this.id = id;

}

showpath(){

let path = [];

clearhilight();

let next = this.position+8;

let divData;

while (next <=64){

if (check(next)){break;};

path.push(next);

next += 8;

}

next = this.position;

while (next%8 !=0){

next += 1;

if (check(next)){break;};

path.push(next);

}

next = this.position-1;

while (next%8 !=0){

if (check(next)){break;};

path.push(next);

next -= 1;

}

next = this.position;

while (next >=9){

next -= 8;

if (check(next)){break;};

path.push(next);

}

hilighed(path);

}

}

This part of the code defines a class named Rook, which represents a rook piece in the game of chess.

The constructor method is used to create a new instance of the Rook class, which takes four parameters: side, piece, position, and id.

side is a string value that represents the color of the rook ("W" for white or "B" for black).

piece is a string value that represents the type of the piece ("R" for rook).

position is an integer value that represents the position of the rook on the board (between 1 and 64, inclusive).

id is a string value that represents the unique identifier of the rook.

The showpath method is used to calculate the legal moves for the rook and highlight them on the board.

This method initializes an empty array named path, which will be used to store the legal moves for the rook. It then calls the clearhilight function to remove any previous highlights on the board.

The method then uses a series of loops to calculate the legal moves for the rook in all four directions (up, down, left, and right) and adds them to the path array.

The check function is called in each iteration of the loop to determine whether the current square is a valid move for the rook. If it returns true, the loop is terminated early and the remaining squares in that direction are not considered.

Finally, the hilighed function is called with the path array as a parameter to highlight the legal moves on the board.

class Knight{

#side;

constructor(side,piece,position,id){

this.#side = side;

this.piece = piece;

this.position = position;

this.id = id;

}

showpath() {

let path = [];

clearhilight();

let row = 8 - Math.floor((this.position - 1) / 8);

let col = String.fromCharCode(65 + ((this.position - 1) % 8));

let moves = [

[2, -1],

[2, 1],

[1, -2],

[1, 2],

[-1, -2],

[-1, 2],

[-2, -1],

[-2, 1]

];

for (let move of moves) {

let newRow = row + move[0];

let newCol = String.fromCharCode(col.charCodeAt(0) + move[1]);

if (newRow < 1 || newRow > 8 || newCol < 'A' || newCol > 'H') {

continue;

}

let next = (8 - newRow) \* 8 + (newCol.charCodeAt(0) - 65) + 1;

if (check(next)){continue;};

path.push(next);

}

hilighed(path);

}

}

This code defines a Knight class. The class has a private field #side which stores the side the knight belongs to (e.g., "white" or "black"), a public field piece which stores the type of piece (i.e., "Knight"), a public field position which stores the current position of the knight on the chessboard, and a public field id which stores a unique identifier for the knight.

The class has a public method showpath() which calculates the valid moves for the knight and highlights them on the chessboard. The method first calls the clearhilight() function to remove any previous highlighting. It then calculates the row and column of the knight's current position on the chessboard using the formula (8 - Math.floor((this.position - 1) / 8)) for the row and String.fromCharCode(65 + ((this.position - 1) % 8)) for the column (where "A" represents the first column and "H" represents the eighth column).

The method defines an array of possible moves for the knight, with each move represented as an array of two integers representing the row and column offsets for the move. The method then iterates through each possible move, calculating the new row and column for each move and checking if the move is valid (i.e., within the bounds of the chessboard and not obstructed by another piece) using the check() function (which is not shown in the provided code snippet).

If the move is valid, the method adds the new position to the path array. Once all possible moves have been checked, the method calls the hilighed() function to highlight the valid moves on the chessboard.

class Bishop{

#side;

constructor(side,piece,position,id){

this.#side = side;

this.piece = piece;

this.position = position;

this.id = id;

}

showpath() {

let path = [];

clearhilight();

let row = Math.floor((this.position - 1) / 8);

let col = (this.position - 1) % 8;

// Check diagonal moves towards top left

let i = row - 1;

let j = col - 1;

while (i >= 0 && j >= 0) {

let next = i \* 8 + j + 1;

if (check(next)){break;};

path.push(next);

i--;

j--;

}

// Check diagonal moves towards top right

i = row - 1;

j = col + 1;

while (i >= 0 && j < 8) {

let next = i \* 8 + j + 1;

if (check(next)){break;};

path.push(next);

i--;

j++;

}

// Check diagonal moves towards bottom left

i = row + 1;

j = col - 1;

while (i < 8 && j >= 0) {

let next = i \* 8 + j + 1;

if (check(next)){break;};

path.push(next);

i++;

j--;

}

// Check diagonal moves towards bottom right

i = row + 1;

j = col + 1;

while (i < 8 && j < 8) {

let next = i \* 8 + j + 1;

if (check(next)){break;};

path.push(next);

i++;

j++;

}

hilighed(path);

}

}

The Bishop class represents the bishop chess piece and contains a constructor and a method called "showpath". The constructor takes in the side (either "white" or "black"), the type of piece ("bishop"), the starting position of the bishop, and an ID number. The showpath method is used to calculate and display the valid moves for the bishop on the game board.

The showpath method first clears any highlighted squares on the game board by calling the clearhilight function. It then calculates the row and column of the bishop's starting position. The method then checks diagonal moves towards the top left, top right, bottom left, and bottom right of the bishop's position by incrementing or decrementing the row and column variables accordingly. For each diagonal move, the method calculates the corresponding square on the game board and checks if that square is a valid move. If it is a valid move, the square is added to the path array.

After all valid diagonal moves have been calculated, the path array is passed to the hilighed function, which highlights the corresponding squares on the game board.

class King{

#side;

constructor(side,piece,position,id){

this.#side = side;

this.piece = piece;

this.position = position;

this.id = id;

}

showpath(){

let path = [];

clearhilight();

let row = Math.floor((this.position - 1) / 8);

let col = (this.position - 1) % 8;

// Check all 8 adjacent squares

for (let i = -1; i <= 1; i++) {

for (let j = -1; j <= 1; j++) {

// Skip the current position

if (i == 0 && j == 0) continue;

let newRow = row + i;

let newCol = col + j;

// Skip invalid positions

if (newRow < 0 || newRow > 7 || newCol < 0 || newCol > 7) continue;

// Get square and highlight it

let next = newRow \* 8 + newCol + 1;

if (check(next)){continue;};

path.push(next);

}

}

hilighed(path);

}

}

This is a code snippet for the showpath method of the King class in a chess game.

The showpath method calculates and highlights all valid moves for the king piece on the chessboard. It first initializes an empty path array and calls the clearhilight function to remove any previously highlighted squares.

The row and col variables are then calculated using the current position of the king on the board.

The method then checks all eight adjacent squares to the current position of the king. This is done using two nested loops that iterate through values of i and j, which represent the relative position of each adjacent square. The continue statement is used to skip the current position.

For each adjacent square that is within the bounds of the chessboard, the method checks if it is a valid move for the king. If it is not a valid move, the square is added to the path array.

Finally, the hilighed function is called, passing in the path array to highlight all valid moves for the king on the chessboard.

class Queen{

#side;

constructor(side,piece,position,id){

this.#side = side;

this.piece = piece;

this.position = position;

this.id = id;

}

showpath(){

let path = [];

clearhilight();

let row = Math.floor((this.position - 1) / 8);

let col = (this.position - 1) % 8;

// Check diagonal moves towards top left

let i = row - 1;

let j = col - 1;

while (i >= 0 && j >= 0) {

let next = i \* 8 + j + 1;

if (check(next)){break;};

path.push(next);

i--;

j--;

}

// Check diagonal moves towards top right

i = row - 1;

j = col + 1;

while (i >= 0 && j < 8) {

let next = i \* 8 + j + 1;

if (check(next)){break;};

path.push(next);

i--;

j++;

}

// Check diagonal moves towards bottom left

i = row + 1;

j = col - 1;

while (i < 8 && j >= 0) {

let next = i \* 8 + j + 1;

if (check(next)){break;};

path.push(next);

i++;

j--;

}

// Check diagonal moves towards bottom right

i = row + 1;

j = col + 1;

while (i < 8 && j < 8) {

let next = i \* 8 + j + 1;

if (check(next)){break;};

path.push(next);

i++;

j++;

}

// Check horizontal and vertical moves

let next = this.position + 8;

let divData;

while (next <=64){

if (check(next)){break;};

path.push(next);

next += 8;

}

next = this.position;

while (next%8 !=0){

next += 1;

if (check(next)){break;};

path.push(next);

}

next = this.position-1;

while (next%8 !=0){

if (check(next)){break;};

path.push(next);

next -= 1;

}

next = this.position;

while (next >=9){

next -= 8;

if (check(next)){break;};

path.push(next);

}

hilighed(path);

}

}

This is a class definition for the chess piece Queen. It has a constructor that initializes the side, piece, position, and id properties of the object.

The showpath() method is used to generate a list of possible moves for the Queen based on its current position on the chessboard. The method starts by creating an empty array called path and clearing any previously highlighted squares with the clearhilight() function. It then calculates the row and column of the Queen's current position on the board.

The method then checks diagonal moves towards top left, top right, bottom left, and bottom right by using while loops that decrement or increment the row and column indices. It checks if the new position is valid with the check() function before adding it to the path array.

The method then checks horizontal and vertical moves by using while loops that increment or decrement the position by 1 or 8 depending on the direction. It also checks if the new position is valid before adding it to the path array.

Finally, the method calls the hilighed() function with the path array to highlight the possible moves on the chessboard.

Overall, this class definition implements the behavior of a Queen piece in a game of chess.

class Pawn {

#side;

constructor(side, piece, position, id) {

this.#side = side;

this.piece = piece;

this.position = position;

this.id = id;

}

showpath() {

let path = [];

clearhilight();

let next = this.position;

//logic for initial 2 square move

if ((this.#side == 'W' && next >= 49 && next <= 56) || (this.#side == 'B' && next >= 9 && next <= 16)) {

if (this.#side == 'W') {

next -= 16;

} else {

next += 16;

}

if (!(check(next))) {

path.push(next);

hilighed(path);

}

if (this.#side == 'W') {

next += 8;

} else {

next -= 8;

}

if (!(check(next))) {

path.push(next);

hilighed(path);

}

return;

}

if (this.#side == 'W') {

next -= 8;

} else {

next += 8;

}

if (!(check(next))) {

path.push(next);

hilighed(path);

}

}

}

This is a class definition for a Pawn in a chess game. The class has a constructor that takes in the side of the pawn ('W' or 'B'), the type of piece ('Pawn'), the starting position of the pawn, and an id. The class also has a method called showpath() which calculates and highlights the legal moves for the pawn on the board.

In the showpath() method, first, the path array is initialized, and then the clearhilight() function is called to clear any previously highlighted squares. The method then checks if the pawn is in its starting position, and if so, it calculates the legal moves for a two-square move. If the pawn is not in its starting position, it simply calculates the legal move for a one-square move.

The legal moves are calculated by determining the position of the square in front of the pawn (either up or down depending on the pawn's side), and checking if that square is occupied by any piece. If the square is unoccupied, it is added to the path array, and the hilighed() function is called to highlight the square on the board.

function hilighed(path){

for(let i =0;i<path.length ; i++){

divData = document.getElementById(''+path[i]);

divData.setAttribute("class","piece-box showpath");

}

}

function check(path){

let next,i,j;

[i,j] = calcij(path - 1);

if (board[i][j]){

return true;

}else{

return false;

}

}

// create a table element

function drawBoard(){

//Add boader with alpha

addBoarder();

// Main content

for (let i=0;i<8;i++){

// Create a row

let divRow = document.createElement("div");

divRow.setAttribute("class","div-row");

//Add boader with number

let divData = document.createElement("div");

divData.setAttribute("class","boarder-box-2");

divData.innerHTML = nums[i];

divRow.appendChild(divData);

for (let j=0;j<8;j++){

// Create a cell

let divData = document.createElement("div");

let boxId = alphas[j] + nums[i];

// Even cells are white cells else black cells

if ((i+j) % 2 == 0){

// Create a class attribute for all white cells

divData.setAttribute("class","piece-box white-box");

divData.setAttribute("id",''+(8\*(i)+(j+1)));

}else{

divData.setAttribute("class","piece-box black-box");

divData.setAttribute("id",''+(8\*(i)+(j+1)));

}

divData.addEventListener("dragover", function(event) {

event.preventDefault();

});

divData.addEventListener("drop", function(event) {

//const data = event.dataTransfer.getData("id");

//console.log(event.target);

dragfun(event.target.id);

//const draggableElement = document.getElementById(data);

//const dropzone = event.target;

//dropzone.appendChild(draggableElement);

});

divData.addEventListener("click", function(event) {

//console.log(this.id);

dragfun(this.id);

});

// Append the cell

divRow.appendChild(divData);

}

//Add boader with number

let divData2 = document.createElement("div");

divData2.setAttribute("class","boarder-box-2");

divData2.innerHTML = nums[i];

divRow.appendChild(divData2);

parentDiv.appendChild(divRow);

}

parentDiv.setAttribute("cellspacing","0");

document.body.appendChild(parentDiv);

addBoarder();

}

function addBoarder(){

let divRow = document.createElement("div");

divRow.setAttribute("class","div-row");

let divData = document.createElement("div");

divData.setAttribute("style","width:40px;height:40px;background-color:rgb(43, 43, 43);");

divRow.appendChild(divData);

for (let i = 0; i <8;i++) {

let divData = document.createElement("div");

divData.setAttribute("class","boarder-box");

divData.innerHTML = alphas[i].toLocaleUpperCase();

divRow.appendChild(divData);

}

let divData2 = document.createElement("div");

divData2.setAttribute("style","width:40px;height:40px;background-color:rgb(43, 43, 43);");

divRow.appendChild(divData2);

parentDiv.appendChild(divRow);

}

This code seems to be a JavaScript implementation of a chess game. It defines a Pawn class with methods for showing the available paths for a pawn and checks for any obstacles in its way. It also includes functions for creating the chessboard and highlighting cells. The drawBoard function creates the HTML for the chessboard with cells identified by alphanumeric coordinates, and also allows for dragging and dropping of pieces. The code suggests that there is a GUI element where users can interact with the chess game, and it appears that parentDiv is the container for this GUI.

Harith Abeysinghe

Check again

I apologize, but I cannot check again without knowing the specific issue you want me to check. Could you please provide more information or context?

Harith Abeysinghe

function hilighed(path){

for(let i =0;i<path.length ; i++){

divData = document.getElementById(''+path[i]);

divData.setAttribute("class","piece-box showpath");

}

}

function check(path){

let next,i,j;

[i,j] = calcij(path - 1);

if (board[i][j]){

return true;

}else{

return false;

}

} explain this part

Sure, I'd be happy to explain this part of the code.

hilighed(path) is a function that takes a list of positions on the board (path) and highlights them by changing the class of the corresponding HTML elements.

The function loops over each position in path, gets the corresponding HTML element using getElementById(), and sets its class to "piece-box showpath". This causes the highlighted squares to be displayed with a different color or style than the other squares on the board.

check(path) is a function that takes a position on the board (path) and checks if there is a piece at that position. The function first converts the position to row and column indices using the calcij() function (which is not shown here), and then checks if there is a piece at that location in the board array. If there is a piece at the specified location, the function returns true, indicating that the move is not valid; otherwise it returns false, indicating that the move is valid.

function drawBoard(){

//Add boader with alpha

addBoarder();

// Main content

for (let i=0;i<8;i++){

// Create a row

let divRow = document.createElement("div");

divRow.setAttribute("class","div-row");

//Add boader with number

let divData = document.createElement("div");

divData.setAttribute("class","boarder-box-2");

divData.innerHTML = nums[i];

divRow.appendChild(divData);

for (let j=0;j<8;j++){

// Create a cell

let divData = document.createElement("div");

let boxId = alphas[j] + nums[i];

// Even cells are white cells else black cells

if ((i+j) % 2 == 0){

// Create a class attribute for all white cells

divData.setAttribute("class","piece-box white-box");

divData.setAttribute("id",''+(8\*(i)+(j+1)));

}else{

divData.setAttribute("class","piece-box black-box");

divData.setAttribute("id",''+(8\*(i)+(j+1)));

}

divData.addEventListener("dragover", function(event) {

event.preventDefault();

});

divData.addEventListener("drop", function(event) {

//const data = event.dataTransfer.getData("id");

//console.log(event.target);

dragfun(event.target.id);

//const draggableElement = document.getElementById(data);

//const dropzone = event.target;

//dropzone.appendChild(draggableElement);

});

divData.addEventListener("click", function(event) {

//console.log(this.id);

dragfun(this.id);

});

// Append the cell

divRow.appendChild(divData);

}

//Add boader with number

let divData2 = document.createElement("div");

divData2.setAttribute("class","boarder-box-2");

divData2.innerHTML = nums[i];

divRow.appendChild(divData2);

parentDiv.appendChild(divRow);

}

parentDiv.setAttribute("cellspacing","0");

document.body.appendChild(parentDiv);

addBoarder();

}

function addBoarder(){

let divRow = document.createElement("div");

divRow.setAttribute("class","div-row");

let divData = document.createElement("div");

divData.setAttribute("style","width:40px;height:40px;background-color:rgb(43, 43, 43);");

divRow.appendChild(divData);

for (let i = 0; i <8;i++) {

let divData = document.createElement("div");

divData.setAttribute("class","boarder-box");

divData.innerHTML = alphas[i].toLocaleUpperCase();

divRow.appendChild(divData);

}

let divData2 = document.createElement("div");

divData2.setAttribute("style","width:40px;height:40px;background-color:rgb(43, 43, 43);");

divRow.appendChild(divData2);

parentDiv.appendChild(divRow);

}

This is a JavaScript code that creates and draws a chessboard on a webpage.

The drawBoard() function is the main function that creates the chessboard. It creates a border with alpha (letters) on the top and bottom of the board and adds the main content of the board. It creates rows using div element and adds border with numbers on the left and right of each row. Then, it creates cells using div element and adds an id attribute to each cell to identify them. Even cells are given a class attribute of "piece-box white-box" and odd cells are given a class attribute of "piece-box black-box". In addition, it adds event listeners to each cell for dragover, drop, and click events.

The addBoarder() function is a helper function used by drawBoard() to add the border with alpha and numbers on the top and bottom of the board. It creates a row using div element and adds a cell with a black background to the left and right of the row. Then, it creates a cell for each letter of the alphabet and adds them to the row. Finally, it adds the same black cell to the left and right of the row.

The parentDiv variable is not shown in the provided code, but it is assumed to be a div element that will contain the chessboard.

function initiateBoard(){

//rooks

let rbl = new Rook('B','RB',1,'RBL');

let rwl = new Rook('W','RW',57,'RWL');

let rbr = new Rook('B','RB',8,'RBR');

let rwr = new Rook('W','RW',64,'RWR');

//knights

let nbl = new Knight('B','NB',2,'NBL');

let nwl = new Knight('W','NW',58,'NWL');

let nbr = new Knight('B','NB',7,'NBR');

let nwr = new Knight('W','NW',63,'NWR');

//bishops

let bbl = new Bishop('B','BB',3,'BBL')

let bwl = new Bishop('W','BW',59,'BWL')

let bbr = new Bishop('B','BB',6,'BBR')

let bwr = new Bishop('W','BW',62,'BWR')

//king and queen

let kb = new King('B','KB',4,'KB')

let kw = new King('W','KW',60,'KW')

let qb = new Queen('B','QB',5,'Qb')

let qw = new Queen('W','QW',61,'QW')

board[0] = [rbl,nbl,bbl,kb,qb,bbr,nbr,rbr]

//black pawns

for (let i= 0;i <8;i++){

let p = new Pawn('B','PB',i+9,'PB'+i)

board[1][i] = p;

}

board[7] = [rwl,nwl,bwl,kw,qw,bwr,nwr,rwr]

//white pawns

for (let i= 0;i <8;i++){

let p = new Pawn('W','PW',i+49,'PW'+i)

board[6][i] = p;

}

}

This function is used to initiate the chess board by creating instances of the different chess pieces and placing them on their starting positions.

The function creates instances of the following chess pieces for both black and white sides:

Rooks (two for each side)

Knights (two for each side)

Bishops (two for each side)

Kings (one for each side)

Queens (one for each side)

Pawns (eight for each side)

Each piece is created using the respective class constructors (Rook, Knight, Bishop, King, Queen, and Pawn) and given a color, an id, a position on the board, and a unique identifier. The pieces are then added to the board array at their respective positions.

function addPieces(){

for (let i = 0; i < 8 ;i++){

for (let j = 0; j < 8 ;j ++){

if (board[i][j]){

//let boxId = alphas[j] + nums[i];

let divData = document.getElementById(''+(8\*(i)+(j+1)));

let img = document.createElement("img");

let p = board[i][j];

img.setAttribute("src",pieces[p.piece]);

img.setAttribute("id",p.id);

img.addEventListener("click", function() {

p.showpath();

drag = p;

});

img.addEventListener("dragstart", function(event) {

p.showpath();

drag = p;

});

divData.appendChild(img);

}

}

}

}

function clearhilight(){

for (let i = 0; i < 8 ;i++){

for (let j = 0; j < 8 ;j ++){

//let boxId = alphas[j] + nums[i];

let divData = document.getElementById((''+(8\*(i)+(j+1))));

if ((i+j) % 2 == 0){

divData.setAttribute("class","piece-box white-box");

}else{

divData.setAttribute("class","piece-box black-box");

}

}

}

}

function dragfun(target){

let previ,prevj,nexti,nextj,next;

[previ,prevj] = calcij(drag.position-1);

next = Number(target);

[nexti,nextj] = calcij(next-1);

console.log(next,board[nexti][nextj],drag.position);

let targetcel = document.getElementById(target);

let bnext = board[nexti][nextj];

/\*if (bnext){

console.log(next,board[nexti][nextj],drag.position);

/\*if (bnext.position != next){

console.log("c");

let child = document.getElementById(board[nexti][nextj].id);

targetcel.removeChild(child);

}\*/

/\*}\*/

drag.position = next;

board[nexti][nextj] = board[previ][prevj];

board[previ][prevj] = 0;

let img = document.getElementById(drag.id);

targetcel.appendChild(img);

let childCount = targetcel.childElementCount;

if (childCount >1){

let firstChild = targetcel.firstElementChild;

firstChild.remove();

}

}

The initiateBoard() function initializes the board by creating and placing chess pieces at their starting positions. The board is represented as a two-dimensional array board, where each element of the array is a reference to a piece object.

The addPieces() function adds chess pieces to the game board displayed on the webpage. For each non-null element of the board array, an HTML img element is created and set with the image of the corresponding chess piece. An event listener is also added to the img element to show the legal moves of the clicked chess piece.

The clearhilight() function clears the highlighted squares on the board.

The dragfun(target) function is a handler function for the drag and drop functionality of the chess pieces. When a piece is dragged and dropped onto a target square, this function updates the position of the chess piece on the board array and moves the corresponding HTML img element to the new square. If the target square already contains a chess piece, the existing piece is removed from the square.

function calcij(num){

i = Math.floor(num/8);

j = num%8;

return [i,j];

}

The calcij function takes a number as input and returns a tuple of two integers representing the corresponding row and column indices of the 8x8 board. The function first calculates the row index by dividing the number by 8 and rounding down to the nearest integer using the Math.floor function. The column index is then obtained by taking the remainder of the number divided by 8 using the modulo operator %. The function then returns the tuple [i,j].