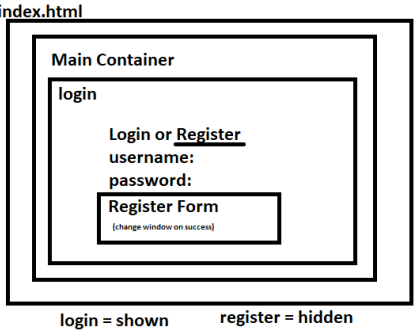
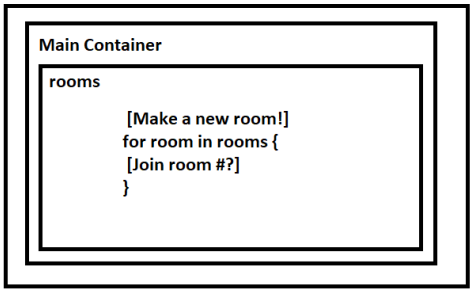
index.html

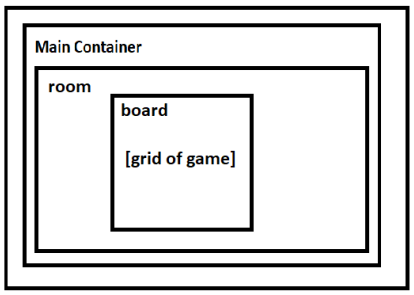


login = shown
rooms = hidden
room = hidden

register = hidden



login = hidden
rooms = shown
room = hidden



login = hidden
rooms = hidden
room = shown

Firebase: Boardgames

```
root - Boardrooms - Lilp???: gameRoom {
                        gameID: 4 letter (show as caps)
                        gameBoard: array[arrays] 8x8
                        player1: {
                                name: username
                                }
                        player2:    ''
                        timeCreated:  a Date()
                        timeLast:  another Date()
                        }
     - Boardlogins - Lilp???: user {
                        username: ''
                        password: ''
                        email:  (use gmail API?)
                        }
```

Chess Model, View, Controller

Model

```
board = [
        [0, ...0]
        ...
        ] 8x8
```

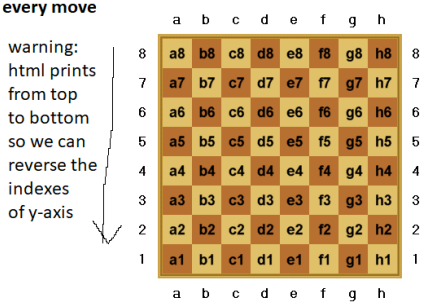
starting positions
0 = empty

11 = w pawn	21 = b pawn
12 = w horse	22 = b horse
13 = w bishop	23 = b bishop
14 = w rook	24 = b rook
15 = w queen	25 = b queen
16 = w king	26 = b king

```
gameState = 'white';
// can also be 'black'
// 'wWin', 'bWin'
// 'wClick', 'bClick' to
      show legal moves
      go back to 'white' or
      'black' if click same
```

View
print board after every move

warning: html prints from top to bottom so we can reverse the indexes of y-axis



html div or table with clickable and hoverable elements (highlight legal moves upon hover? or click? or both)

Controller
generate a grid (the board) using datakeys = 'a8' etc down to 'h1' (for loops nested) (store a-h as an array)

onclick events for class .boardTile
onclick events will read datakey, compare it to array (parse string and return array indices: [0,0]
each piece type has legal moves

Math.floor(pieceValue/10) = 2nd digit (piece color)
pieceValue%10 = 1st digit (piece rank)

```
function legalMoves(pieceValue, board) {
      // determine legal moves, in board boundaries
      return arrayLegal;  // can be 'a8' or array of arrays
}
```

Check and Checkmate

Check
Warns you if your king is currently under threat
Your next move cannot leave your king under threat

function to check if king is threatened

```
function kingThreat(color, board) {
    // in the given board, is the king already under
    threat?

    // brute force - check 64 positions

    // having additional datastructures (array of objs
    which are pieces) check up to 16 pieces

    return true or false;
}
```

(should be called under legalMoves to cancel out self-check inducing moves)

Checkmate
You are under check by your opponent's last move
All legalMoves result in check, therefore legalMoves actually returns an empty array[]

Not being able to move should be checkmate

Pawns:
can move forward
can capture diagonally
turns into another piece at board top

Castling:
king and rook must've not moved (need a datastruct)
no pieces in between (0's)
legal move marking: king, 2 spaces towards one rook to activate

Room Logic

What do you need to enter a room
- -a name that matches a logged in name
- -maybe passwords are not necessary
  (no, it is necessary to prevent duplicates unless you hack) (can also handle this using the Firebase key)
- -can enter 4 letter room code to spectate game or take a seat

What you can do in a room (logistically)
- -take a seat (your name gets logged to room object, your name must match player in order to click the board at your turn
- -if you are a spectator you can't click board maybe add a chatbox
- -if you are playing and it's your turn you can click via game controller (if turnInvalid() break;)

Taking a seat can also depend on Firebase Key

Login Logic

Register

```
boardlogin.push({
    // player login info
    username: ,password: , email: ,created:
    ,lastLogin:
})
```

Login

```
for (key in response
if (username === response[key].username && pw) {
    // local data (refreshes clears this)
    gamerName = username
    gamerKey = key
}
```

Logout

location.reload() ?

Legal Moves

Pawn

Space in front if empty

Space to diagonal-front if enemy

(activates promotion if moved to space at end of board)

Knight

spaces at +/-1,+/-2
and +/- 2,+/-1
(8 different combinations)
(legal if on board and if space not occupied by ally)

bishop

diagonal spaces, all unoccupied spaces in continuous line and including first enemy but excluding first ally

for loop, [+/-i, +/-i] (4 paths away from bishop, check until first illegal (or last legal) move)

rook

straight spaces, all unoccupied spaces until first enemy (inclusive) or first ally (exclusive)

for loop, [0,+/-i]
another for loop (not nested) [+/-i, 0]

queen

for loops of both bishop and rook, can probably call these submethods

king

check the 8 spaces around it
[+/-1, +/-1] (4 spaces)
[+/-1, 0] (2 spaces)
[0, +/-1] (2 spaces)

king position also checks for castling

```
any piece legalMove(space, board) {

legalArray = [];

// determine piece on the space
// if your piece continue, else return [];

// knowing piece and space
// run method on piece and space
// example: knight -> returns array of
   legal moves as array of strings
// for move of array, check if king is
under check for next board:
isCheck(color,board) -> splice from
array if move leads to self-check

}
```