Project Report

Title: Online Chess Game

Web Application: https://onlinechess-game.herokuapp.com

Project By

Amey Thakur & Mega Satish

ABSTRACT

We proposed to build an online chess game for single-player as well as multiplayer in nodejs and deploy it on Heroku. An online chess game is developed in compliance with Human-Machine Interaction principles. The web app has a simple and attractive design. The theme of the game can also be customised by the player. The chessboard's background may be changed to suit the user's preferences. Users can play against other players in multiplayer mode, which features a chatbox for real-time conversation between players. Aside from that, chosen blocks will be highlighted for easier sight, and a sound will be heard after each move to confirm the move. While playing, the user learns about the potential movement of the pieces. The browser will notify you if an opponent leaves the game in the middle of a round in multiple player mode.

LIST OF FIGURES

LIST OF ABBREVIATIONS

Acronym	Abbreviation
GUI	Graphical User Interface
UI	User Interface
UX	User Experience
I/O	Input/ Output
CPU	Central Processing Unit
ECMA	European Computer Manufacturers Association Script
API	Application Programming Interface
TCP	Transmission Control Protocol
IP	Internet Protocol
CLI	Command Line Interface

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

1.1 Brief description of the project

The project “Online Chess Game” implements a classic version of Chess with a Graphical User Interface (GUI). The Chess game follows the basic rules of chess, and all the chess pieces only move according to valid moves for that piece. It is played on an 8x8 checkerboard, with a dark square in each player's lower-left corner. The website was developed using Nodejs and deployed on Heroku. This is an online chess game that users from remote locations can play and discuss on the website through the chat window. The user can play in single mode and multiplayer mode as well. There is also an option to change the layout of the chessboard as the player would like. While playing, the player gets to know about his current position and the other position in which he can move his pieces.

1.2 What is a chess game?

Chess is a game for 2 players each of whom moves 16 pieces according to fixed rules across a checkerboard and tries to checkmate the opponent's king. Our project implements the chess game with a graphical user interface. The chess game follows the basic rules of chess and all the chess pieces only move according to valid moves for that piece.

1.3 What Are The Chess Pieces?

The chess pieces are what you move on a chessboard when playing a game of chess. There are six different types of chess pieces. Each side starts with 16 pieces: eight pawns, two bishops, two knights, two rooks, one queen, and one king.

The Pawn

When a game begins, each side starts with eight pawns. White's pawns are located in the second rank, while Black's pawns are located in the seventh rank. The pawn is the least powerful piece and is worth one point. If it is a pawn's first move, it can move forward one or two squares. If a pawn has already moved, then it can move forward just one square at a time. It attacks (or captures) each square

diagonally to the left or right. In the following diagram, the pawn has just moved from the e2-square to the e4-square and attacks the squares d5 and f5.

The Bishop

Each side starts with two bishops, one on a light square and one on a dark square. When a game begins, White's bishops are located on c1 and f1, while Black's bishops are located on c8 and f8. The bishop is considered a minor piece (like a knight) and is worth three points. A bishop can move diagonally as many squares as it likes, as long as it is not blocked by its own pieces or an occupied square. An easy way to remember how a bishop can move is that it moves like an "X" shape. It can capture an enemy piece by moving to the occupied square where the piece is located.

The Knight

Each side starts with two knights—a king's knight and a queen's knight. When a game starts, White's knights are located on b1 and g1, while Black's knights are located on b8 and g8. The knight is considered a minor piece (like a bishop) and is worth three points. The knight is the only piece in chess that can jump over another piece! It moves one square left or right horizontally and then two squares up or down vertically, OR it moves two squares left or right horizontally and then one square up or down vertically—in other words, the knight moves in an "L-shape." The knight can capture only what it lands on, not what it jumps over!

The Rook

Each side starts with two rooks, one on the queenside and one on the kingside. All four rooks are located in the corners of the board. White's rooks start the game on a1 and h1, while Black's rooks are located on a8 and h8. The rook is considered a major piece (like the queen) and is worth five points. It can move as many squares as it likes left or right horizontally, or as many squares as it likes up or down vertically (as long as it isn't blocked by other pieces).

The Queen

The queen is the most powerful chess piece! When a game begins, each side starts with one queen. The white queen is located on d1, while the black queen is located on d8. The queen is considered a major piece (like a rook) and is worth nine points. It can move as many squares as it likes left or right horizontally, or as many squares as it likes up or down vertically (like a rook). The queen can also move as many squares as it likes diagonally (like a bishop).

The King

The king is the most important chess piece. The goal of a game of chess is to checkmate the king. When a game starts, each side has one king. White's king is located on e1, while Black's king starts on e8. The king is not a very powerful piece, as it can only move (or capture) one square in any direction. When a king is attacked, it is called a "check."

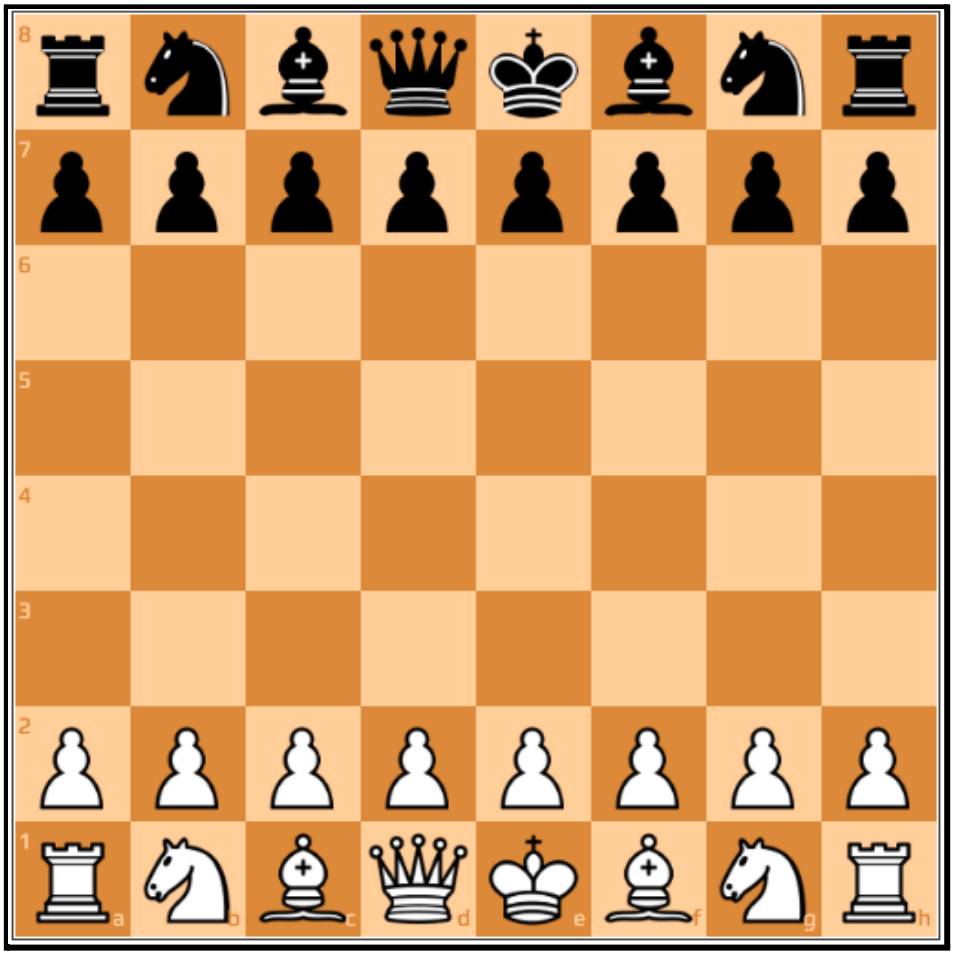


Figure 1: Chess Board

CHAPTER 2
PROBLEM STATEMENT

Chess is a game where the battle of minds takes place between two people. It is a game of strategy where two people play with each other's minds. It's a board game that requires patience, concentration, intuition, perseverance, etc. Chess is a mind game that involves a lot of thinking and time. It requires prediction and problem-solving skills. We are living in a world that is connected despite being in different locations. So we have the ability to play and challenge other people with games. The project focuses on a chess game that anyone can play online with someone or by themselves. The main goal is to allow the gamer to have a good experience of the play but also connect with the opponent player and learn and discuss chess moves.

CHAPTER 3
METHODOLOGY

3.1 HMI principles implemented in the project

1. **Place Users at the Centre**

- As always, the first UI design principle is to focus on people (or, the "user" as we all say). A good user interface is easy and natural to use, avoids confusing the user, and does what the user needs. Learning about human-centred design will help you achieve the right mindset for the best interfaces and focus on people first, and design second.

2. **Strive for Clarity**

- The purpose of the user interface is to allow the user to interact with the website or application (or, more generally in broader design, any product). Avoid anything that confuses people or doesn't help them interact.

3. **Minimise Actions and Steps Per Screen**

- Streamline tasks and actions so they can be done in as few steps as possible. Each screen should have one primary focus. Keep the primary action front and centre and move secondary actions deeper on a page or give them lighter visual weight and the right typography.

4. **Aim for Simplicity**

- Classics exist for a reason; they're timeless and never go out of style, though they do benefit from modern touches. A user interface should be simple and elegant.

5. **Be Consistent**

- Consistency creates familiarity, and familiar interfaces are naturally more usable. Consistent design reduces friction for the user. A consistent design is predictable. Predictable design means it's easy to understand how to use functions without instruction. Not only should UI design be consistent

internally, but externally as well. A design system is a great way to ensure consistency in UI design.

6. **Your Goal: Make Your User Interface Design Invisible**

- Don't draw attention to your user interface. A great UI allows people to use the product without friction, not spend time figuring out how to interact with the product.

7. **Provide Useful Feedback**

- Feedback can be visual, audio (the ding of a new message alert), or sense of touch. Every action should have feedback to indicate whether the action was successful or not.
- Feedback helps to answer questions in four areas:
    - Location: You are here.
    - Status: What's going on? Is it still going on?
    - Future status: What's next?
    - Outcomes & Results: Hey, what happened?
- Hovering over a navigation item that then changes colour indicates an item is clickable. Buttons should look like buttons. Feedback lets the user know if they're doing the right thing (or the wrong thing).

8. **Reduce Cognitive Load**

- Many of these UI design principles serve to reduce cognitive load for users. Basically, don't make users think (also a useful UX design principle as well).

9. **Make It Accessible**

- UI designs need to take into account accessibility issues. Online, this often means ensuring the visibly impaired can access and use the product.

10. **Flexible**

- Create a UI that will work and look great across multiple platforms. It may have to be tweaked depending on the form factor of a device and its

operating system (Android and iOS, for example), but it should be flexible enough to work on anything.

11. Visual Structure

- Keep a consistent visual structure to create familiarity and relieve user anxiety by making them feel at home. A few elements to focus on include a visual hierarchy with the most important things made obvious, colour scheme, consistent navigation, re-use elements, and creating a visual order using grids.

12. Dialogues Should Result in Closure

- Actions should have a beginning, middle, and end (with feedback at each step). For example, when making an online purchase we move from browsing and product selection to the checkout and then finally confirmed that the purchase is completed.

13. Provide a Clear Next Step

- Include a clear next step a user can take after an interaction. That could be as simple as a "back to top" click at the end of a long blog post or a pointer to more information. Help the user achieve their goals with the next step.

3.2 NodeJs

- Node.js is an open-source and cross-platform JavaScript runtime environment. It is a popular tool for almost any kind of project!
- Node.js runs the V8 JavaScript engine, the core of Google Chrome, outside of the browser. This allows Node.js to be very performant.
- A Node.js app runs in a single process, without creating a new thread for every request. Node.js provides a set of asynchronous I/O primitives in its standard library that prevent JavaScript code from blocking and generally, libraries in Node.js are written using non-blocking paradigms, making blocking behaviour the exception rather than the norm.
- When Node.js performs an I/O operation, like reading from the network, accessing a database or the filesystem, instead of blocking the thread and

wasting CPU cycles waiting, Node.js will resume the operations when the response comes back.

- This allows Node.js to handle thousands of concurrent connections with a single server without introducing the burden of managing thread concurrency, which could be a significant source of bugs.
- Node.js has a unique advantage because millions of frontend developers that write JavaScript for the browser are now able to write the server-side code in addition to the client-side code without the need to learn a completely different language.
- In Node.js the new ECMAScript standards can be used without problems, as you don't have to wait for all your users to update their browsers - you are in charge of deciding which ECMAScript version to use by changing the Node.js version, and you can also enable specific experimental features by running Node.js with flags.

3.3 Socket Programming

- A socket is a communication connection point (endpoint) that you can name and address in a network. Socket programming shows how to use socket APIs to establish communication links between remote and local processes.
- The processes that use a socket can reside on the same system or different systems on different networks. Sockets are useful for both stand-alone and network applications. Sockets allow you to exchange information between processes on the same machine or across a network, distribute work to the most efficient machine, and they easily allow access to centralised data. Socket application program interfaces (APIs) are the network standard for TCP/IP. A wide range of operating systems support socket APIs. i5/OS sockets support multiple transports and networking protocols. Socket system functions and the socket network functions are thread-safe.

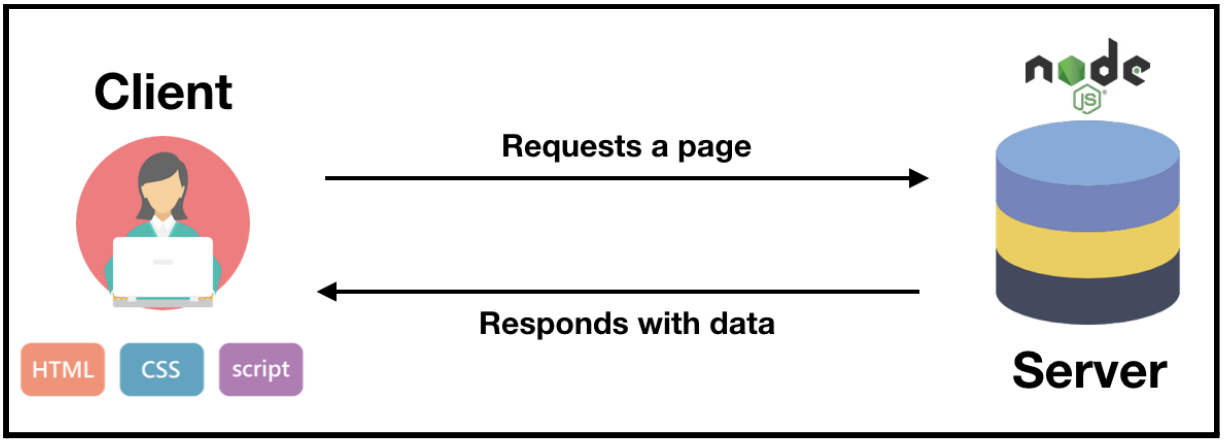


Figure 2: Socket Programming with Nodejs

3.4 Heroku

The project is hosted on Heroku which is a cloud Platform as a container-based Service (PaaS). Heroku is used by developers to launch, manage, and grow contemporary programs. Heroku is an open-source software platform for machine learning and data science that makes it simple to develop and publish attractive, bespoke web apps. The benefit of web apps is that they are platform agnostic and may be operated by anybody with an Internet connection. Their code is run on a back-end server, which processes incoming requests and answers using a common protocol that all browsers can understand.

Deployment:

1. Install dependencies of node js using packages.json
2. Build and run the app locally using the command heroku local web
3. Create git and commit the changes
4. Deploy application on Heroku using CLI
5. To open the app, type heroku open.

CHAPTER 4

IMPLEMENTATION

GitHub Repository: https://github.com/Amey-Thakur/ONLINE-CHESS-GAME

4.1 Index.html

```
<!DOCTYPE html>
<html>

<head>
  <title>AMEY</title>
  <link rel="icon" href="./img/icon.jpg">
  <link rel="stylesheet" type="text/css" href="./css/semantic.min.css">
  <link rel="stylesheet" href="./css/chessboard-1.0.0.min.css">
  <link rel="stylesheet" href="./css/styles.css">
</head>

<body>
  <audio id="myAudio">
    <source src="./mp3/soundMove.mp3" type="audio/mpeg">
    Your browser does not support the audio element.
  </audio>
  <audio id="messageTone">
    <source src="./mp3/insight.mp3" type="audio/mpeg">
    Your browser does not support the audio element.
  </audio>
  <!-- Navbar -->
  <div>
     <div style="margin: 0; border-bottom: 4px solid gray; padding: 3px 0; " class="ui
secondary menu">
       <img src="./img/icon.png" style="width:80px;height:80px;">
       <h2 style="text-decoration: underline;">CHESS GAME</h2>
       <div class="right menu">
```

```
                    <div style="margin-top:20px; height: 40px; padding-right: 20px;
margin-right: 15px;"
              class="ui labeled button" tabindex="0">
              <div class="ui button">
                <i class="user icon"></i> #PLAYERS
              </div>
              <a class="ui basic label">
                <span id="players">0</span>
              </a>
            </div>
                     <div style="margin-top:20px; height: 40px; padding-left: 20px;
margin-right: 15px;"
              class="ui labeled button" tabindex="0">
              <div class="ui button">
                <i class="star icon"></i> #ROOMS
              </div>
              <a class="ui basic label">
                <span id="rooms">0</span>
              </a>
            </div>
          </div>
        </div>
      </div>
      <div>
        <div id="gameMode">
                 <h1 style="text-align: center; margin: 10px; font-size: 35px;"> GAME
MODE</h1>
          <div style="text-align:center;">
                    <button class="game ui black button " id="singlePlayer">SINGLE
PLAYER</button>
          </div>
          <div style="text-align:center;">
```

```
                    <button class="game ui black button " id="multiPlayer">MULTI
PLAYER</button>
        </div>
      </div>
      <div id="joinFormDiv" style="display: none;">
        <form id="joinForm">
                <h1 style="text-align: center; margin: 10px; font-size: 35px;">START
GAME</h1>
          <div style="text-align:center;">
                          <input class="formInput" type="text" placeholder="Name"
style="padding: 10px;">
          </div>
          <div style="text-align:center;">
                          <input class="formInput" type="text" placeholder="Room"
style="padding: 10px;">
          </div>
            <div id="roomDropdownP" style="text-align:center;  height: 50px; margin:
10px; padding: 0 4px; ">
            <div id="roomDropdown" class="ui fluid search selection dropdown"
              style="border: 1px solid gray; width: 300px; margin:auto;">
              <input type="hidden" name="country">
              <i class="dropdown icon"></i>
              <div class="default text">SELECT ROOM</div>
              <div class="menu" id="dropRooms">
                                  <!-- <div class="item" data-value="af"><i class="icon
star"></i>Afghanistan</div>
                                      <div class="item" data-value="ar"><i class="icon
star"></i>Argentina</div> -->
              </div>
            </div>
          </div>
          <div style="text-align:center;">
            <button class="game ui black button " id="joinButton">JOIN</button>
```

```
        </div>
        <div style="text-align:center;">
          <p id="message"></p>
        </div>
      </form>
    </div>
    <!-- /Input Form -->
    <div>
      <!-- Chess Board -->
      <div id="chessGame" style="display: none;">
        <!-- Color Schemes -->
        <div style="text-align: center; margin: 10px;">
          <button id="grey_board" class="ui button black color_b">GREY</button>
                                <button id="orange_board" class="ui button grey
color_b">ORANGE</button>
                                 <button id="green_board" class="ui button grey
color_b">GREEN</button>
          <button id="blue_board" class="ui button grey color_b">BLUE</button>
        </div>
        <!-- Status and PGN -->
        <div id="statusPGN" style="text-align: center; display: none;">
          <div>
            <label>
              <h3><strong>STATUS</strong></h3>
            </label>
            <div id="status">YOUR TURN</div>
          </div>
          <div>
            <label>
              <h3><strong>HISTORY</strong></h3>
            </label>
                  <div id="pgn" style="overflow: auto; white-space: nowrap; width:
500px; margin: auto;"></div>
```

```
            </div>
          </div>
              <div id="myBoard" style="width: 569px; margin: auto; margin-top: 10px;
margin-bottom: 10px;"></div>
          <div style="text-align: center; margin-bottom: 20px;">
                    <a href="/" class="ui button black" style="width: 569px;">LEAVE
GAME</a>
          </div>
        </div>
      </div>
    </div>
    <!-- Chatting window -->
    <div id="chat"
       style=" background-color: white; display: none; text-align: right; position: fixed;
bottom: 0; right: 0; width: 400px; margin-right: 10px; border: 2px solid black;">
      <div class="ui button grey" style="border-radius: 0; width: 100%; padding: 15px;
font-size: 16px;"
        id="messageBox">
        MESSAGES
      </div>
      <div id="chatBox" style="display: none; padding: 12px;">
            <div id="chatContent" style="height: 240px; overflow-y: auto; word-break:
break-all; ">
          <!-- <div class="myMessage">Hello</div>
          <div class="youMessage">his</div> -->
        </div>
        <form style="margin-bottom: 0;" class="ui form">
          <div style="display: flex; justify-content: space-around;">
                      <input class="form-control " id="inputMessage" type="text"
placeholder="Enter a Message"
              style="margin-right: 10px;">
            <button class="ui black button" id="send">SEND</button>
          </div>
```

```
        </form>
      </div>
   </div>
   <div
        style="background-color: black; color: white; margin-top:auto; padding: 15px;
text-align: center; font-size: 15px;">
       <i>"Avoid the crowd. Do your own thinking independently. Be the chess player,
not the chess piece"</i></div>
   <script src="https://code.jquery.com/jquery-3.1.1.min.js"
           integrity="sha256-hVVnYaiADRTO2PzUGmuLJr8BLUSjGIZsDYGmIJLv2b8="
crossorigin="anonymous"></script>
   <script src="./js/semantic.min.js"></script>
   <script src="/socket.io/socket.io.js"></script>
   <script src="./js/chessboard-1.0.0.min.js"></script>
        <script   src="https://cdnjs.cloudflare.com/ajax/libs/chess.js/0.10.2/chess.js"
integrity="sha384-s3XgLpvmHyscVpijnseAmye819Ee3yaGa8NxstkJVyA6nuDFjt59u
1QvuEl/mecz"
      crossorigin="anonymous"></script>
   <script src="./js/app.js"></script>
</body>
</html>
```

4.2 Style.css

```
html {
   margin: 0;
   padding: 0;
   height: 100%;
}

body {
   margin: 0;
   padding: 0;
```

```
    height: 100%;
    overflow: visible;
    display:flex;
    flex-direction:column;
}
 #singlePlayer, #multiPlayer, #joinButton{
    margin: 10px;
    width: 300px;
    padding: 20px;
    font-size: 22px;
 }

 #gameMode, #joinFormDiv{
    padding: 10px;
    margin-top: 8%;
 }

 .black-3c85d {
    background-color: #E1E1E1;
    color: #FFFFFF;
}

.white-1e1d7 {
    background-color: #FFFFFF;
    color: #E1E1E1;
}

#grey_board, #orange_board, #green_board, #blue_board{
    margin: 6px;
    width: 130px;
}
.formInput {
    border: 1px solid gray;
```

```
    height: 50px;
    border-radius: 4px;
    width: 300px;
    margin: 10px;
    padding: 0 4px;
}

.myMessage{
    text-align: right;
    padding: 8px;
    background-color: #E0E0E0;
    font-size: 17px;
    margin-bottom: 10px;
    margin-right: 7px;
    clear:both;
    float : right;
    border-radius: 5px;
    border-left: 12px solid transparent;
    border-right: 12px solid transparent;
}

.youMessage{
    text-align: left;
    padding: 8px;
    background-color: #616161;
    color: white;
    font-size: 17px;
    margin-bottom: 10px;
    clear:both;
    border-radius: 5px;
    float : left;
}
```

4.3 App.js

```
const formEl = document.querySelectorAll('#joinForm > div > input')
const joinButtonEl = document.querySelector('#joinButton')
const messageEl = document.querySelector('#message')
const statusEl = document.querySelector('#status')
const ChatEl = document.querySelector('#chat')
const sendButtonEl = document.querySelector('#send')
const roomsListEl = document.getElementById('roomsList');
const myAudioEl = document.getElementById('myAudio');
const singlePlayerEl = document.getElementById('singlePlayer');
const multiPlayerEl = document.getElementById('multiPlayer');
const totalRoomsEl = document.getElementById('rooms')
const totalPlayersEl = document.getElementById('players')
const chatContentEl = document.getElementById('chatContent')
var config = {};
var board = null;
var game = new Chess()
var turnt = 0;

// initializing semantic UI dropdown
$('.ui.dropdown')
  .dropdown();

// function for defining onchange on dropdown menus
$("#roomDropdown").dropdown({
  onChange: function (val) {
    console.log(val)
    console.log('running the function')
    formEl[1].value = val
  }
});
```

```
function onDragStart2(source, piece, position, orientation) {
    // do not pick up pieces if the game is over
    if (game.game_over()) {
        if (game.in_draw()) {
            alert('Game Draw!!');
        }
        else if (game.in_checkmate())
            if (turnt === 1) {
                alert('You won the game!!');
            } else {
                alert('You lost!!');
            }
        return false
    }

    // only pick up pieces for White
    if (piece.search(/^b/) !== -1) return false
}

function makeRandomMove() {
    var possibleMoves = game.moves()

    // game over
    if (possibleMoves.length === 0) {
        return;
    }

    var randomIdx = Math.floor(Math.random() * possibleMoves.length)
    game.move(possibleMoves[randomIdx]);
    myAudioEl.play();
    turnt = 1 - turnt;
    board.position(game.fen());
}
```

```
function onDrop2(source, target) {
  // see if the move is legal
  var move = game.move({
    from: source,
    to: target,
    promotion: 'q' // NOTE: always promote to a queen for example simplicity
  })
  myAudioEl.play();
  // illegal move
  if (move === null) return 'snapback'
  turnt = 1 - turnt;
  // make random legal move for black
  window.setTimeout(makeRandomMove, 250)
}
// update the board position after the piece snap
// for castling, en passant, pawn promotion
function onSnapEnd2() {
  board.position(game.fen())
}

singlePlayerEl.addEventListener('click', (e) => {
  e.preventDefault();
  document.getElementById('gameMode').style.display = "none";
  document.querySelector('#chessGame').style.display = null;
  config = {
    draggable: true,
    position: 'start',
    onDragStart: onDragStart2,
    onDrop: onDrop2,
    onSnapEnd: onSnapEnd2
  }
  board = Chessboard('myBoard', config);
})
```

```
//Connection will be established after webpage is refreshed
const socket = io()
//Triggers after a piece is dropped on the board
function onDrop(source, target) {
   //emits event after piece is dropped
   var room = formEl[1].value;
   myAudioEl.play();
   socket.emit('Dropped', { source, target, room })
}
//Update Status Event
socket.on('updateEvent', ({ status, fen, pgn }) => {
   statusEl.textContent = status
   fenEl.textContent = fen
   pgnEl.textContent = pgn
})
socket.on('printing', (fen) => {
   console.log(fen)
})
//Catch Display event
socket.on('DisplayBoard', (fenString, userId, pgn) => {
   console.log(fenString)
   //This is to be done initially only
   if (userId != undefined) {
      messageEl.textContent = 'Match Started!! Best of Luck...'
      if (socket.id == userId) {
         config.orientation = 'black'
      }
      document.getElementById('joinFormDiv').style.display = "none";
      document.querySelector('#chessGame').style.display = null
      ChatEl.style.display = null
      document.getElementById('statusPGN').style.display = null
   }
```

```
    config.position = fenString
    board = ChessBoard('myBoard', config)
    document.getElementById('pgn').textContent = pgn
})
//To turn off dragging
socket.on('Dragging', id => {
    if (socket.id != id) {
        config.draggable = true;
    } else {
        config.draggable = false;
    }
})
//To Update Status Element
socket.on('updateStatus', (turn) => {
    if (board.orientation().includes(turn)) {
        statusEl.textContent = "Your turn"
    }
    else {
        statusEl.textContent = "Opponent's turn"
    }
})
//If in check
socket.on('inCheck', turn => {
    if (board.orientation().includes(turn)) {
        statusEl.textContent = "You are in Check!!"
    }
    else {
        statusEl.textContent = "Opponent is in Check!!"
    }
})
//If win or draw
socket.on('gameOver', (turn, win) => {
    config.draggable = false;
```

```
    if (win) {
        if (board.orientation().includes(turn)) {
            statusEl.textContent = "You lost, better luck next time :)"
        }
        else {
            statusEl.textContent = "Congratulations, you won!!"
        }
    }
    else {
        statusEl.value = 'Game Draw'
    }
})
//Client disconnected in between
socket.on('disconnectedStatus', () => {
    alert('Opponent left the game!!')
    messageEl.textContent = 'Opponent left the game!!'
})
//Receiving a message
socket.on('receiveMessage', (user, message) => {
    var chatContentEl = document.getElementById('chatContent')
    //Create a div element for using bootstrap
    chatContentEl.scrollTop = chatContentEl.scrollHeight;
    var divEl = document.createElement('div')
    if (formEl[0].value == user) {
        divEl.classList.add('myMessage');
        divEl.textContent = message;
    }
    else {
        divEl.classList.add('youMessage');
        divEl.textContent = message;
        document.getElementById('messageTone').play();
    }
    var style = window.getComputedStyle(document.getElementById('chatBox'));
```

```
    if (style.display === 'none') {
        document.getElementById('chatBox').style.display = 'block';
    }
    chatContentEl.appendChild(divEl);
    divEl.focus();
    divEl.scrollIntoView();
})
//Rooms List update
socket.on('roomsList', (rooms) => {
    // roomsListEl.innerHTML = null;
    // console.log('Rooms List event triggered!! ',  rooms);
    totalRoomsEl.innerHTML = rooms.length
    var dropRooms = document.getElementById('dropRooms')
    while (dropRooms.firstChild) {
        dropRooms.removeChild(dropRooms.firstChild)
    }
    // added event listener to each room
    rooms.forEach(x => {
        var roomEl = document.createElement('div')
        roomEl.setAttribute('class', 'item')

        roomEl.setAttribute('data-value', x)
        roomEl.textContent = x;
        dropRooms.appendChild(roomEl)
    })
})
socket.on('updateTotalUsers', totalUsers => {
    console.log('event listened')
    totalPlayersEl.innerHTML = totalUsers;
})
//Message will be sent only after you click the button
sendButtonEl.addEventListener('click', (e) => {
    e.preventDefault()
```

```
    var message = document.querySelector('#inputMessage').value
    var user = formEl[0].value
    var room = formEl[1].value
    document.querySelector('#inputMessage').value = ''
    document.querySelector('#inputMessage').focus()
    socket.emit('sendMessage', { user, room, message })
})
//Connect clients only after they click Join
joinButtonEl.addEventListener('click', (e) => {
    e.preventDefault()
    var user = formEl[0].value, room = formEl[1].value
    if (!user || !room) {
        messageEl.textContent = "Input fields can't be empty!"
    }
    else {
        joinButtonEl.setAttribute("disabled", "disabled");
        formEl[0].setAttribute("disabled", "disabled")
        document.querySelector('#roomDropdownP').style.display = 'none';
        formEl[1].setAttribute("disabled", "disabled")
        //Now Let's try to join it in room // If users more than 2 we will
        socket.emit('joinRoom', { user, room }, (error) => {
            messageEl.textContent = error
            if (alert(error)) {
                window.location.reload()
            }
            else    //to reload even if negative confirmation
                window.location.reload();
        })
        messageEl.textContent = "Waiting for other player to join"
    }
})

multiPlayerEl.addEventListener('click', (e) => {
```

```
    e.preventDefault();
    document.getElementById('joinFormDiv').style.display = "block";
    document.getElementById('gameMode').style.display = "none";
    //Server will create a game and clients will play it
    //Clients just have to diaplay the game
    var board = ChessBoard('myBoard')
    config = {
        draggable: false,   //Initially
        position: 'start',
        onDrop: onDrop,
        orientation: 'white'
    }
})
const applyColorScheme = (black, white) => {
    const blackEl = document.querySelectorAll('.black-3c85d');
    for (var i = 0; i < blackEl.length; i++) {
        blackEl[i].style.backgroundColor = black;
        blackEl[i].style.color = white;
    }
    const whiteEl = document.querySelectorAll('.white-1e1d7');
    for (var i = 0; i < whiteEl.length; i++) {
        whiteEl[i].style.backgroundColor = white;
        whiteEl[i].style.color = black;
    }
}
//For removing class from all buttons
const removeClass = () => {
    const buttonEl = document.querySelectorAll('.color_b');
    for (var i = 0; i < buttonEl.length; i++) {
        buttonEl[i].classList.remove('black');
        buttonEl[i].classList.remove('grey');
    }
}
```

```
// Colour Buttons
document.getElementById('grey_board').addEventListener('click', e => {
    e.preventDefault();
    removeClass();
    document.getElementById('grey_board').classList.add('black');
    document.getElementById('orange_board').classList.add('grey');
    document.getElementById('green_board').classList.add('grey');
    document.getElementById('blue_board').classList.add('grey');
    applyColorScheme("#E1E1E1", "#FFFFFF");
})

document.getElementById('orange_board').addEventListener('click', e => {
    e.preventDefault();
    removeClass();
    document.getElementById('grey_board').classList.add('grey');
    document.getElementById('orange_board').classList.add('black');
    document.getElementById('green_board').classList.add('grey');
    document.getElementById('blue_board').classList.add('grey');
    applyColorScheme("#D18B47", "#FFCE9E");
})

document.getElementById('green_board').addEventListener('click', e => {
    e.preventDefault();
    removeClass();
    document.getElementById('grey_board').classList.add('grey');
    document.getElementById('orange_board').classList.add('grey');
    document.getElementById('green_board').classList.add('black');
    document.getElementById('blue_board').classList.add('grey');
    applyColorScheme("#58AC8A", "#FFFFFF");
})

document.getElementById('blue_board').addEventListener('click', e => {
    e.preventDefault();
```

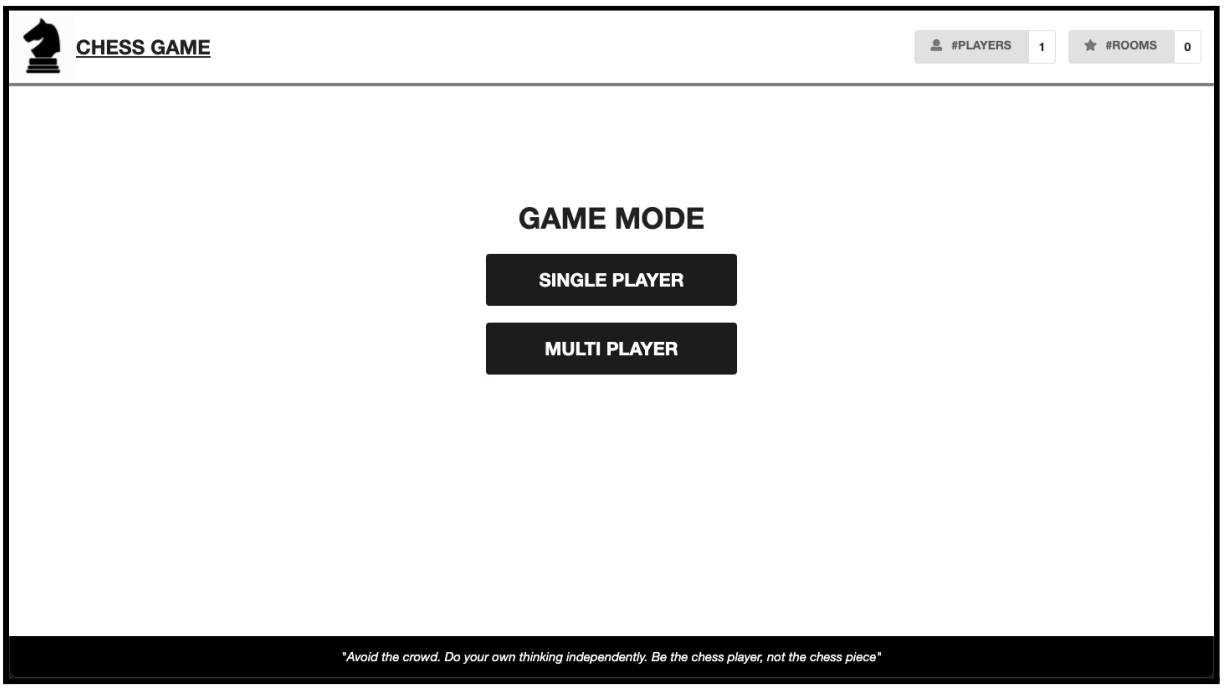
```
    removeClass();
    document.getElementById('grey_board').classList.add('grey');
    document.getElementById('orange_board').classList.add('grey');
    document.getElementById('green_board').classList.add('grey');
    document.getElementById('blue_board').classList.add('black');
    applyColorScheme("#727FA2", "#C3C6BE");
})

// Messages Modal
document.getElementById('messageBox').addEventListener('click', e => {
    e.preventDefault();
    var style = window.getComputedStyle(document.getElementById('chatBox'));
    if (style.display === 'none') {
        document.getElementById('chatBox').style.display = 'block';
    } else {
        document.getElementById('chatBox').style.display = 'none';
    }
})
```

CHAPTER 5

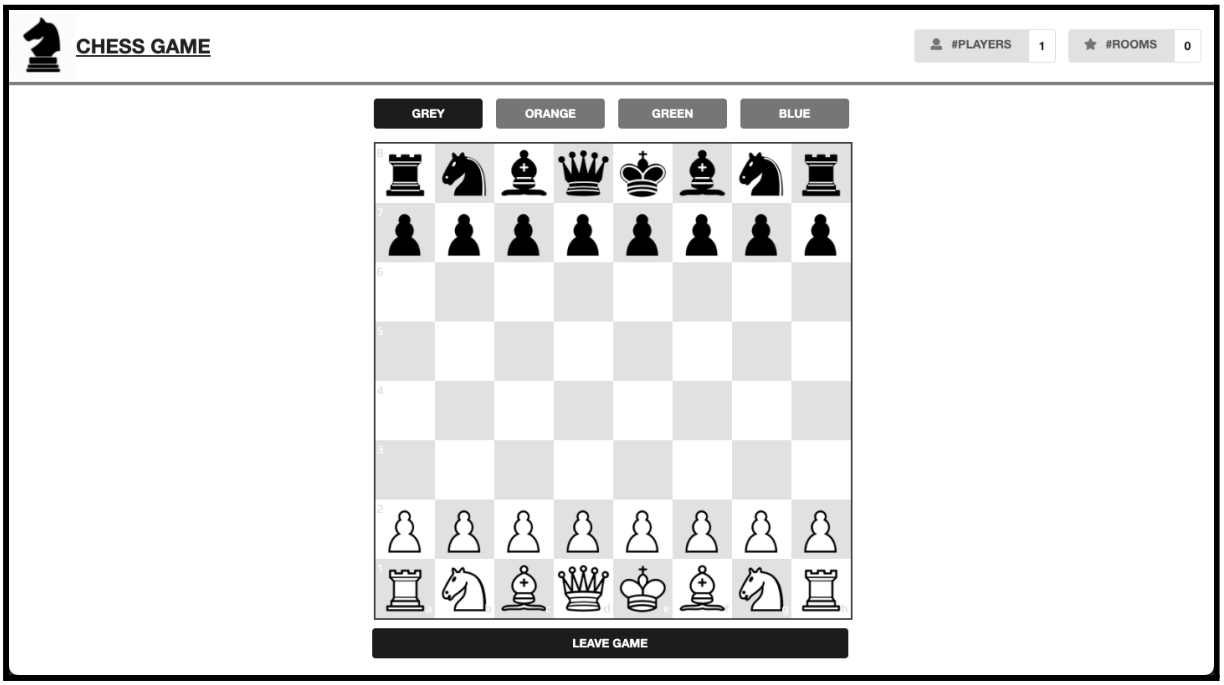
RESULTS

Web Application: https://onlinechess-game.herokuapp.com

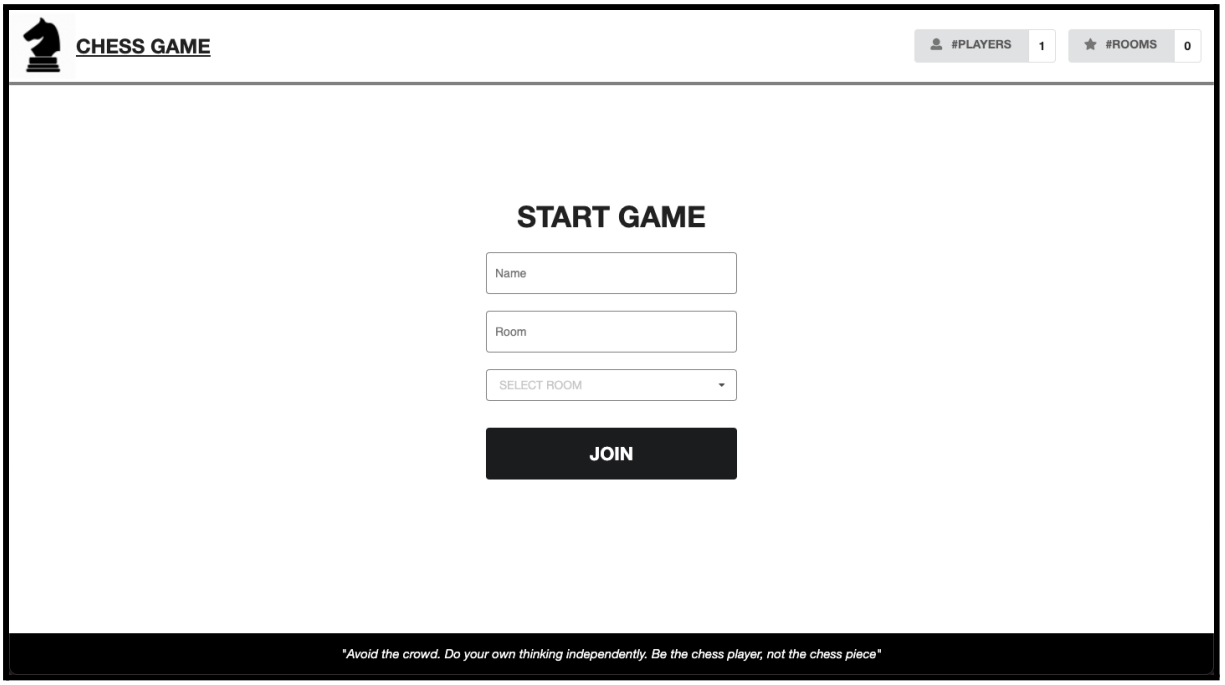
Home Page

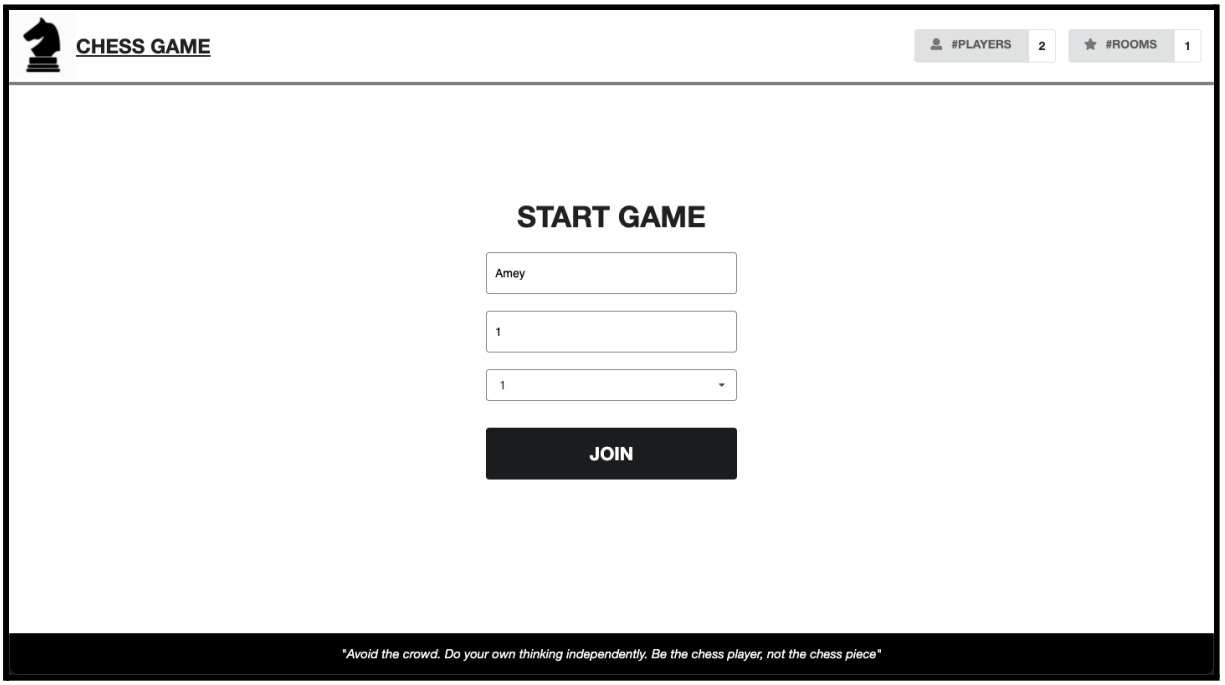


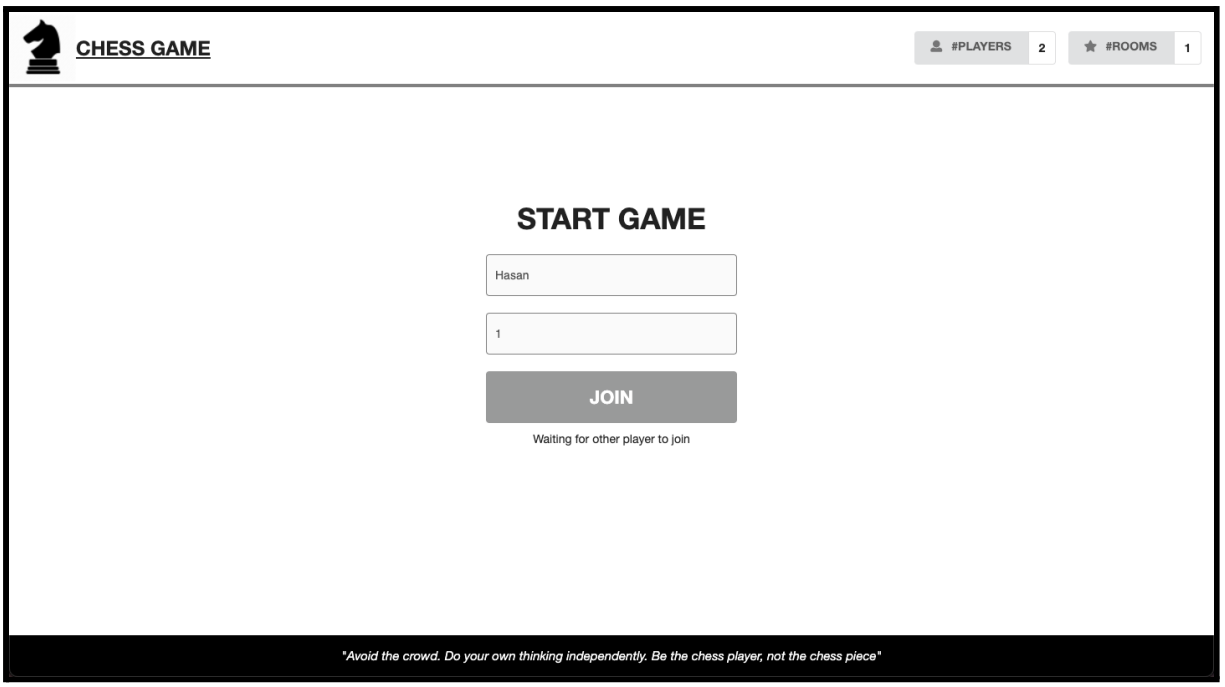
Single Player Mode

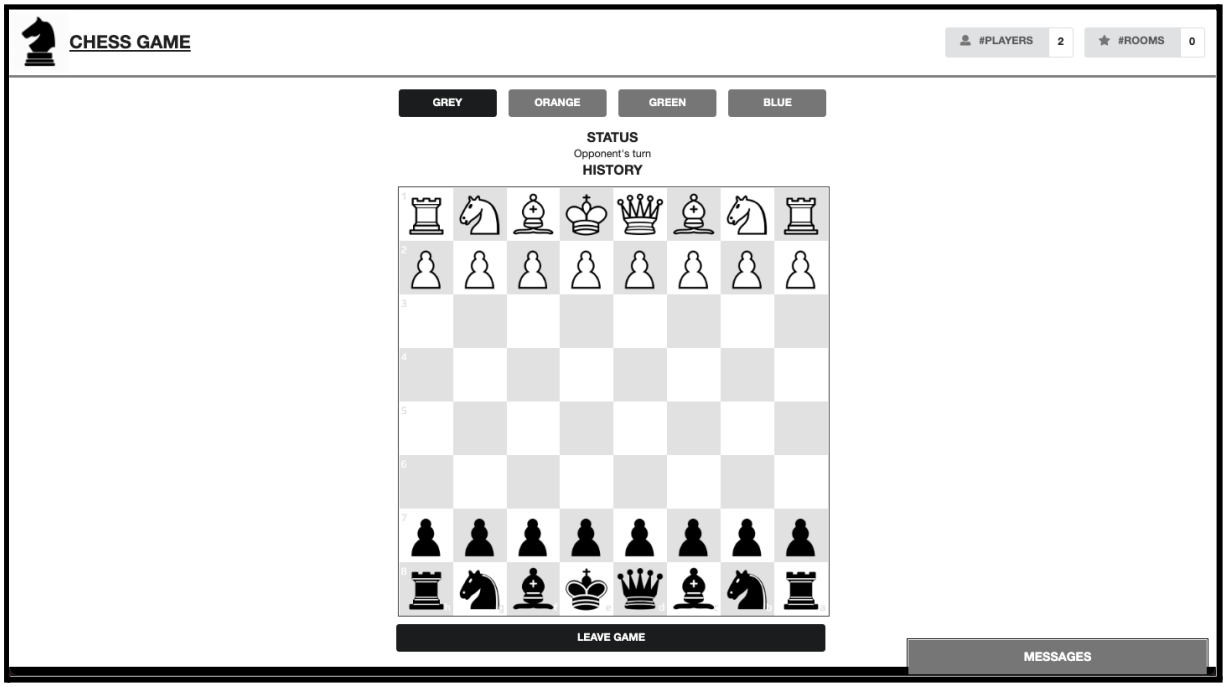


Multiplayer Mode

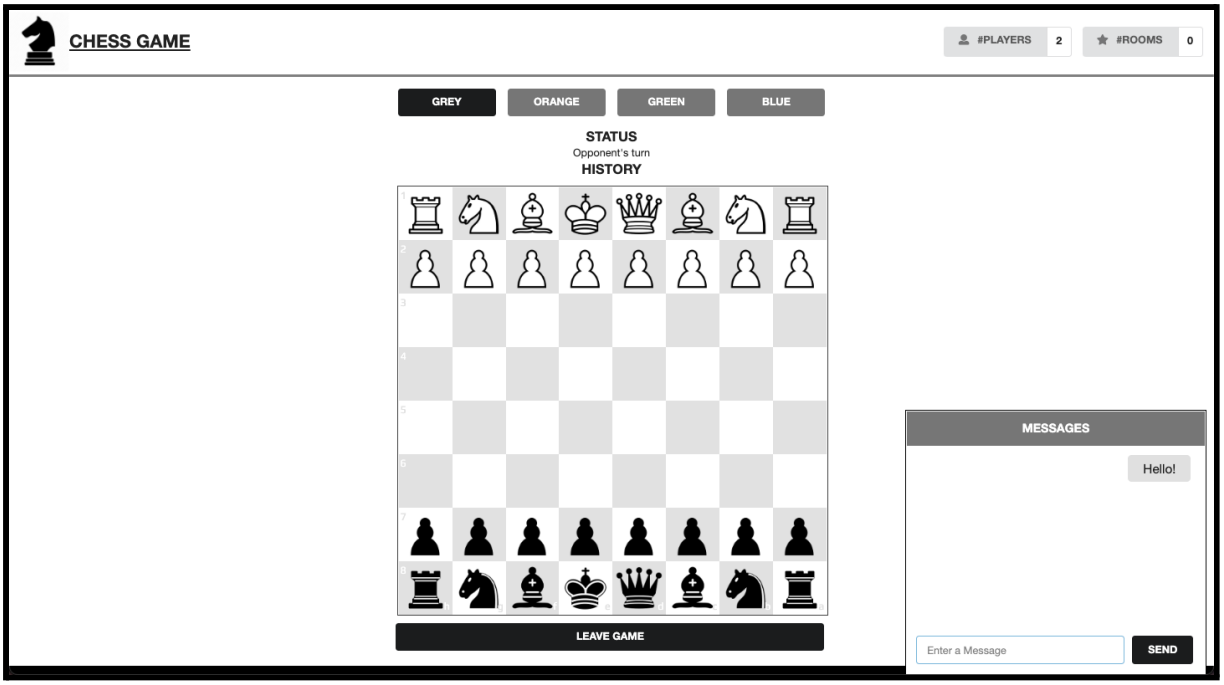


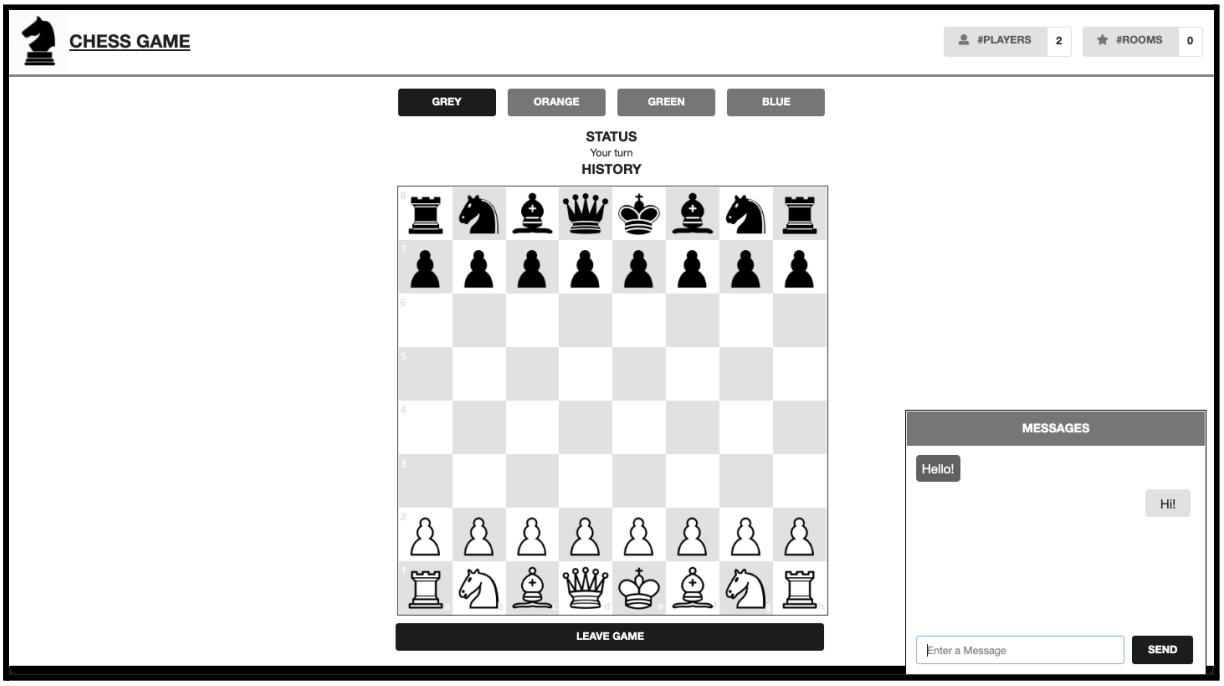



CHESS GAME
#PLAYERS 2
#ROOMS 1
START GAME
Hasan
1
JOIN
Waiting for other player to join
"Avoid the crowd. Do your own thinking independently. Be the chess player, not the chess piece"


CHESS GAME
#PLAYERS 2
#ROOMS 0
GREY
ORANGE
GREEN
BLUE
STATUS
Opponent's turn
HISTORY
LEAVE GAME
MESSAGES

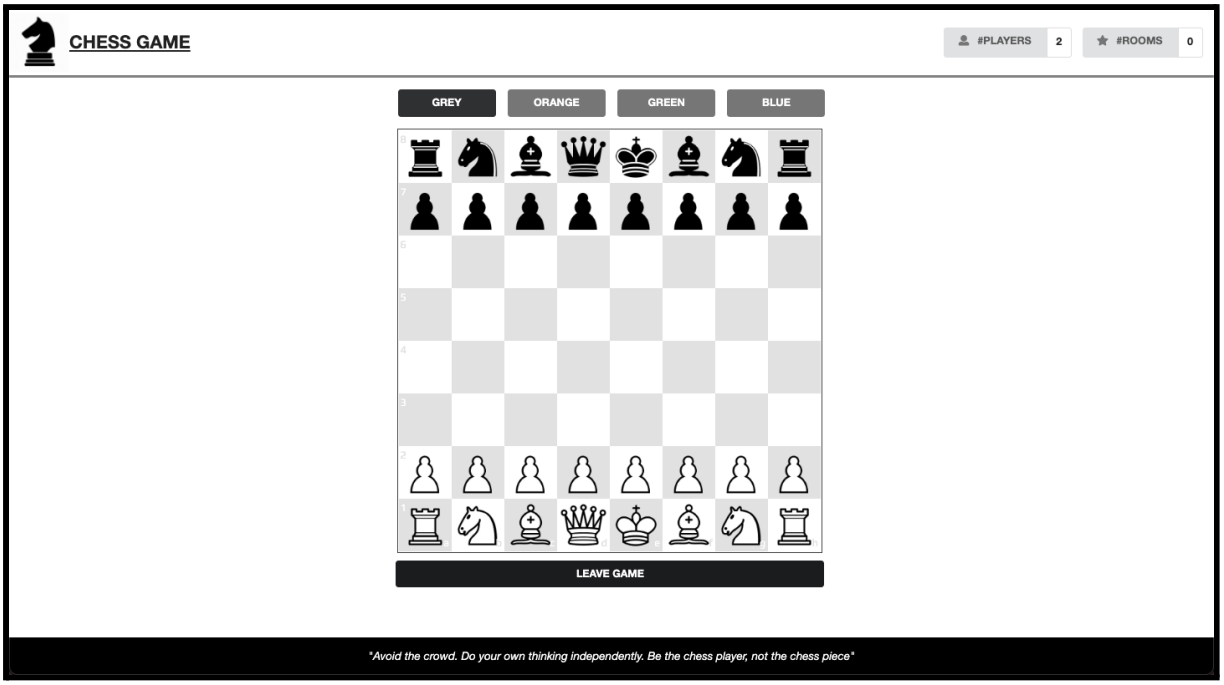
Chat window for players to send message to each other



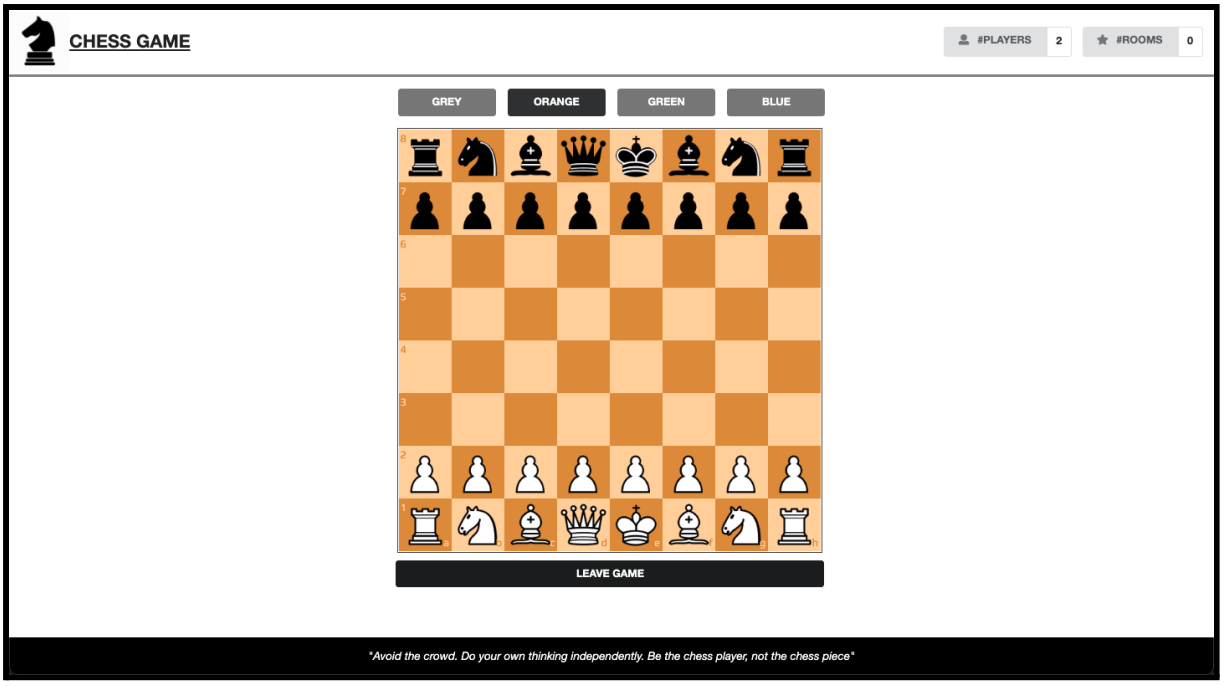


Different Colour Themes:

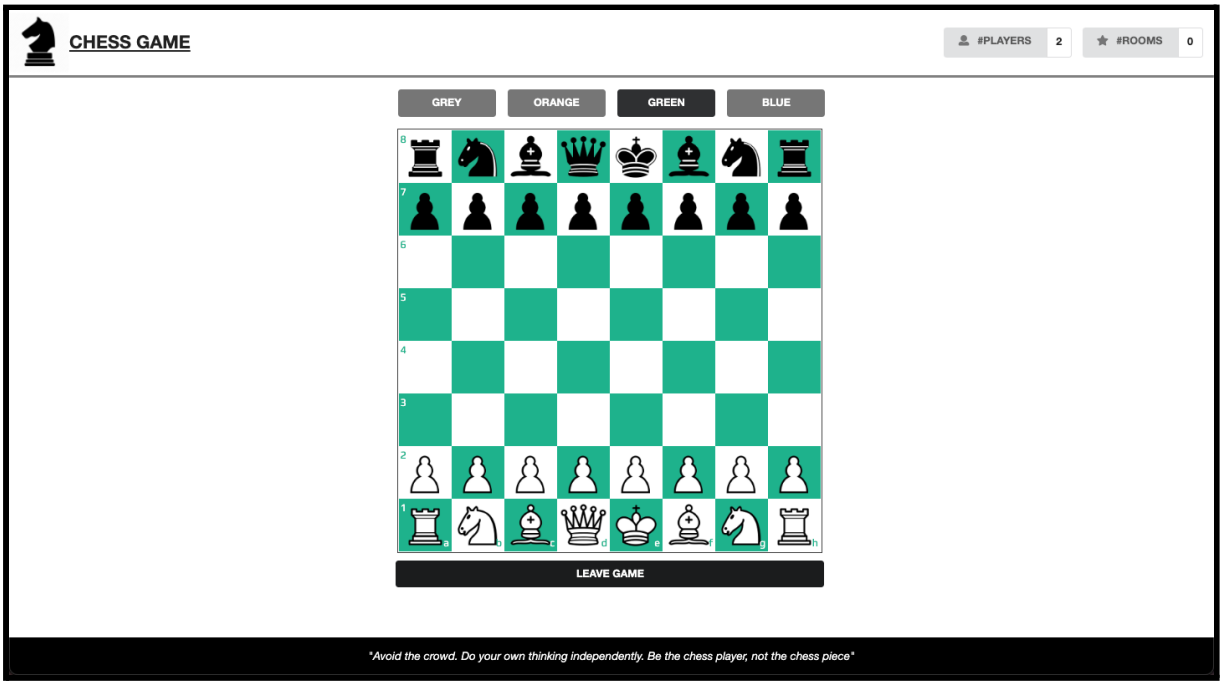
Grey



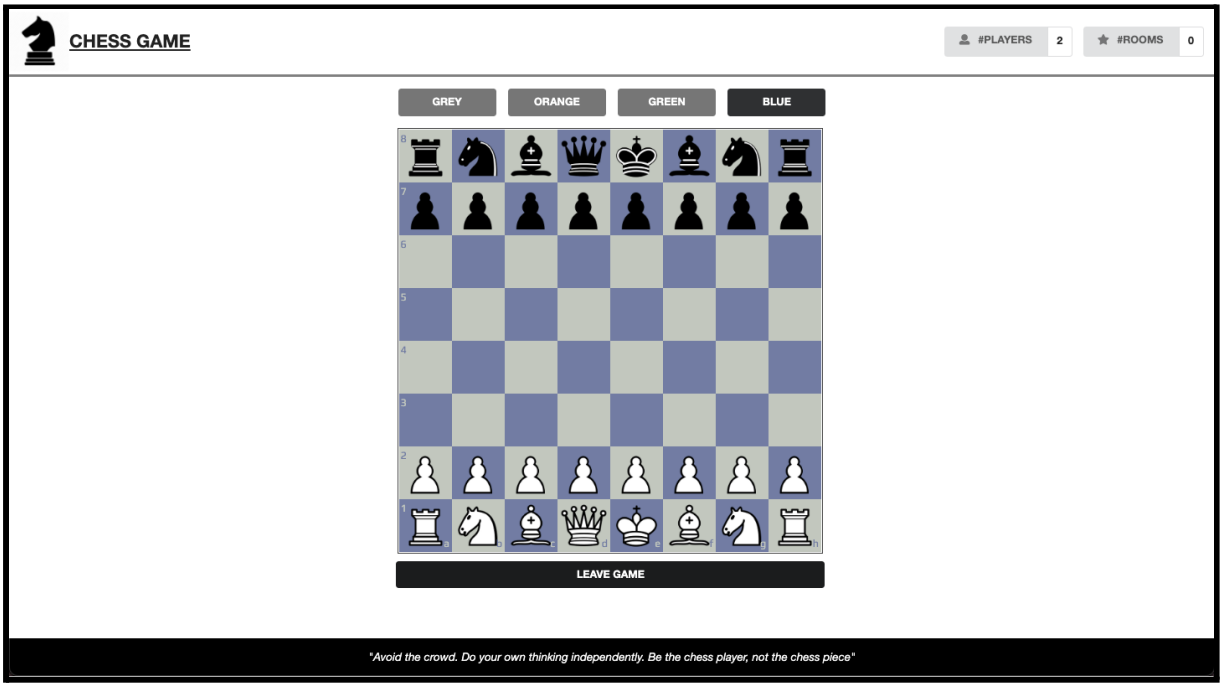
Orange



Green


CHESS GAME
#PLAYERS 2
#ROOMS 0
GREY
ORANGE
GREEN
BLUE
LEAVE GAME
Avoid the crowd. Do your own thinking independently. Be the chess player, not the chess piece

Blue


CHESS GAME
#PLAYERS 2
#ROOMS 0
GREY
ORANGE
GREEN
BLUE
LEAVE GAME
Avoid the crowd. Do your own thinking independently. Be the chess player, not the chess piece

CHAPTER 6
CONCLUSION

Through the proposed system, we can draw the conclusion that the online chess game is designed by keeping in mind the Human-Machine Interaction principles. The web application is simple and allows the user to play chess. Two modes are available: the first one is single-player and the second one is multiplayer. In the case of multiplayer, the user can communicate with the opponent through a chat window. The design of the website is classic and elegant with fewer colours and objects on it and the goal is very clear to the user: playing chess. The user has the choice to change the theme of the game as well. While playing, the user gets to know about the possible movement of the pieces. For multiple player mode, if the opponent leaves the game in the middle, the browser notifies that the opponent has left the game. The user is well informed of everything that is happening and can take decisions without thinking too much.

The web application developed using Nodejs was deployed on the Heroku platform and can be accessed by everyone. Furthermore, for future work, we can add more features and also make a tutorial portal for new beginners with tips on how to play.

REFERENCES

1. https://www.chess.com/terms/chess-pieces
2. https://nodejs.org/en/docs
3. https://devcenter.heroku.com/categories/reference
4. https://devcenter.heroku.com/articles/getting-started-with-nodejs