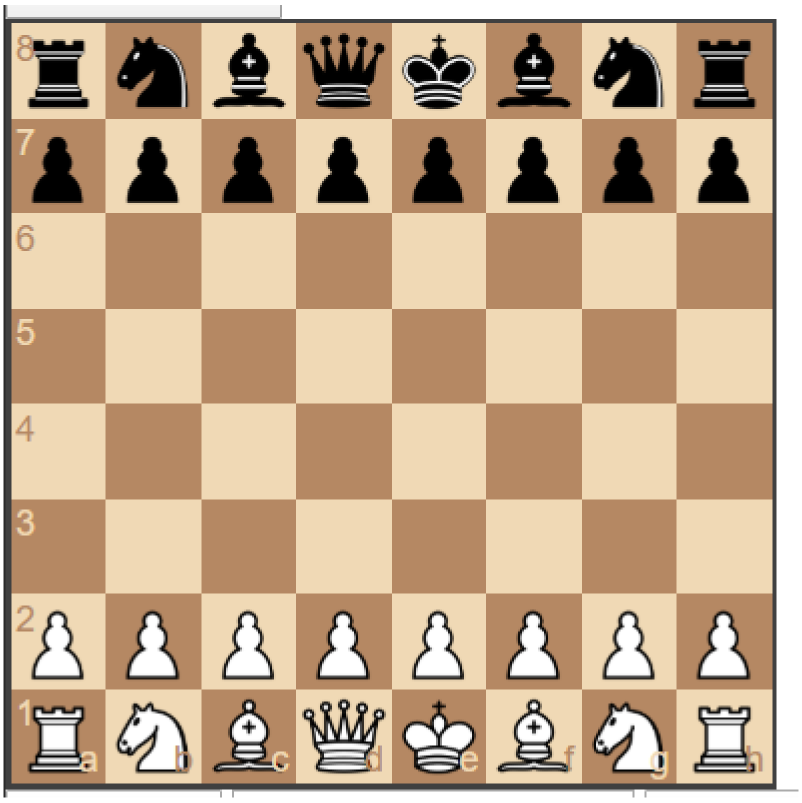
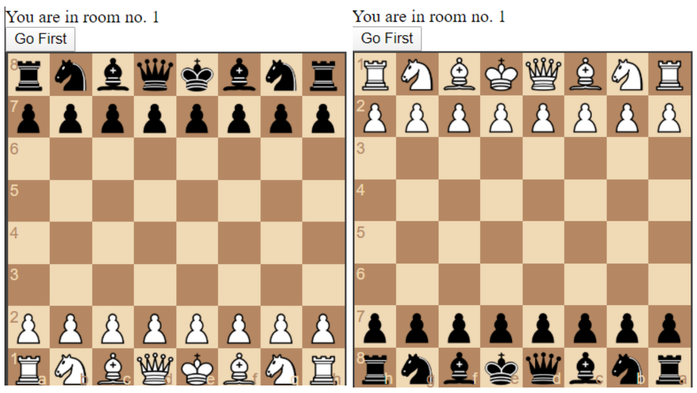
Soft 355 - Coursework – Online Chess

Functionality

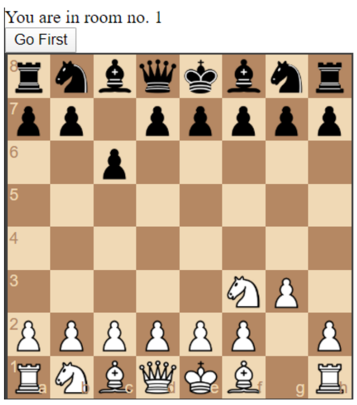
The application is an online chess game allowing for players to play against each other. When the user visits the site, they are presented and are put into a random room with another player.

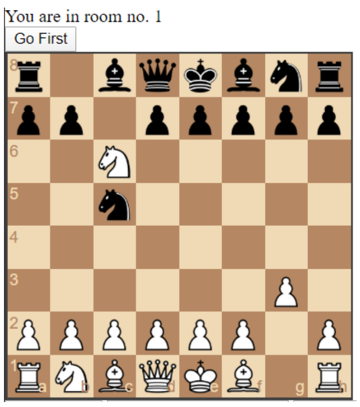


One of the players requests to go first and if the other player accepts then the boards are set to the correct orientation. The players then take it in turns to make moves, the board is locked if it is not there move so they can't cheat. You can also only move your team to prevent cheating as well.



A player can make a move by dragging a piece to the desired spot. This can only be a legal move if not the piece snaps back to its square and not action is taken. If a piece takes another piece then that piece is removed automatically.

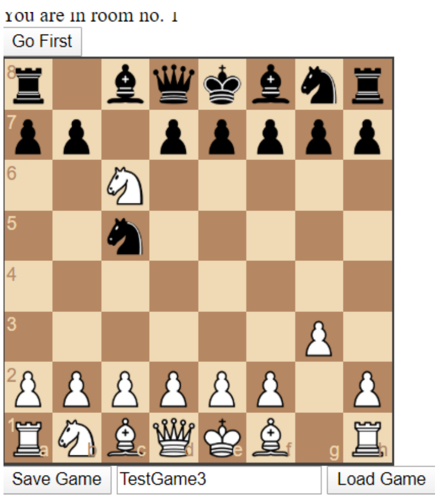


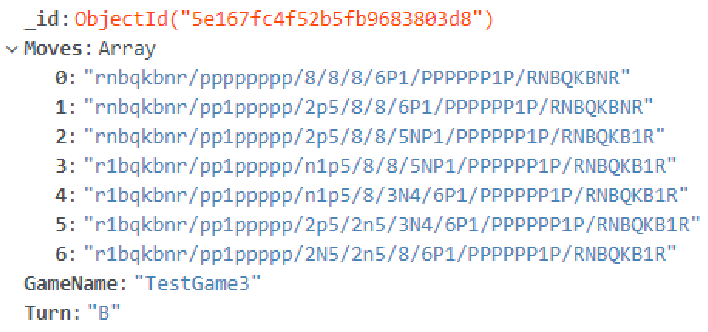


After each move the board is checked to make sure the king has not been removed. If the king has been removed, then the player who made the move wins and the game is paused. A new game can be started at any point by clicking the rest board button. This also requires the permission of both players.

If the king has not been taken, then the game continues, and the correct action is taken, and the game continues. Unless a player concedes.

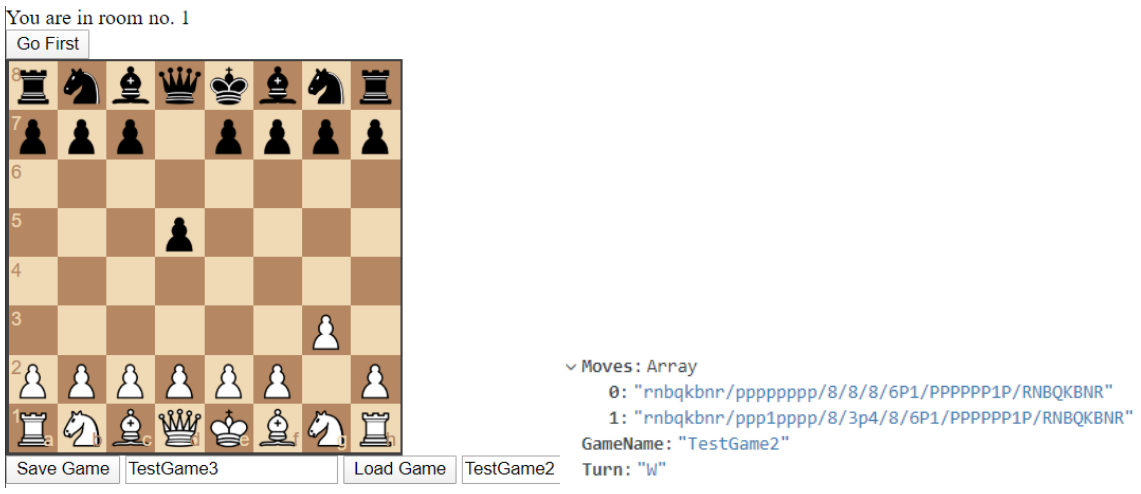
The users also have the ability to save the game state as it is, and it can be loaded again later. This requires the user to input a name for the game. This is then checked to be unique before it is successful added to the database.





```
_id: ObjectId("5e167fc4f52b5fb9683803d8")
⌄ Moves: Array
    0: "rnbqkbnr/pppppppp/8/8/8/6P1/PPPPPP1P/RNBQKBNR"
    1: "rnbqkbnr/pp1ppppp/2p5/8/8/6P1/PPPPPP1P/RNBQKBNR"
    2: "rnbqkbnr/pp1ppppp/2p5/8/8/5NP1/PPPPPP1P/RNBQKB1R"
    3: "r1bqkbnr/pp1ppppp/n1p5/8/8/5NP1/PPPPPP1P/RNBQKB1R"
    4: "r1bqkbnr/pp1ppppp/n1p5/8/3N4/6P1/PPPPPP1P/RNBQKB1R"
    5: "r1bqkbnr/pp1ppppp/2p5/2n5/3N4/6P1/PPPPPP1P/RNBQKB1R"
    6: "r1bqkbnr/pp1ppppp/2N5/2n5/8/6P1/PPPPPP1P/RNBQKB1R"
  GameName: "TestGame3"
  Turn: "B"
```

This can be loaded later by imputing the same game and sets up the board on the last move. This also needs to be confirmed by the opponent.

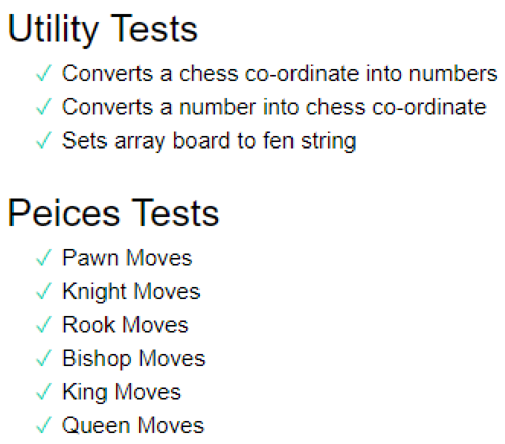

You are in room no. 1

```
⌄ Moves: Array
    0: "rnbqkbnr/pppppppp/8/8/8/6P1/PPPPPP1P/RNBQKBNR"
    1: "rnbqkbnr/ppp1pppp/8/3p4/8/6P1/PPPPPP1P/RNBQKBNR"
  GameName: "TestGame2"
  Turn: "W"
```

The server used in the project is a node.js server. This oversees the communication between players and the database. It also loads all of the resources necessary for the program to work. I have used web sockets form socket.io to communicate between the server and the players. They are organised in rooms so only two people are playing at a time and it doesn't get confusing or breaks.

The database is a non sql database I am utilising mongoose and mongoDB being hosted online. This allows for the games to be stored or loaded at will.

For testing I am using mocha and chai test and assertion. These help me make sure that my program is doing what is meant to do as I make changes to the program. This is useful in identifying if something in the back has broken by accident.


```
Utility Tests
  ✓ Converts a chess co-ordinate into numbers
  ✓ Converts a number into chess co-ordinate
  ✓ Sets array board to fen string

Peices Tests
  ✓ Pawn Moves
  ✓ Knight Moves
  ✓ Rook Moves
  ✓ Bishop Moves
  ✓ King Moves
  ✓ Queen Moves
```

Requirements

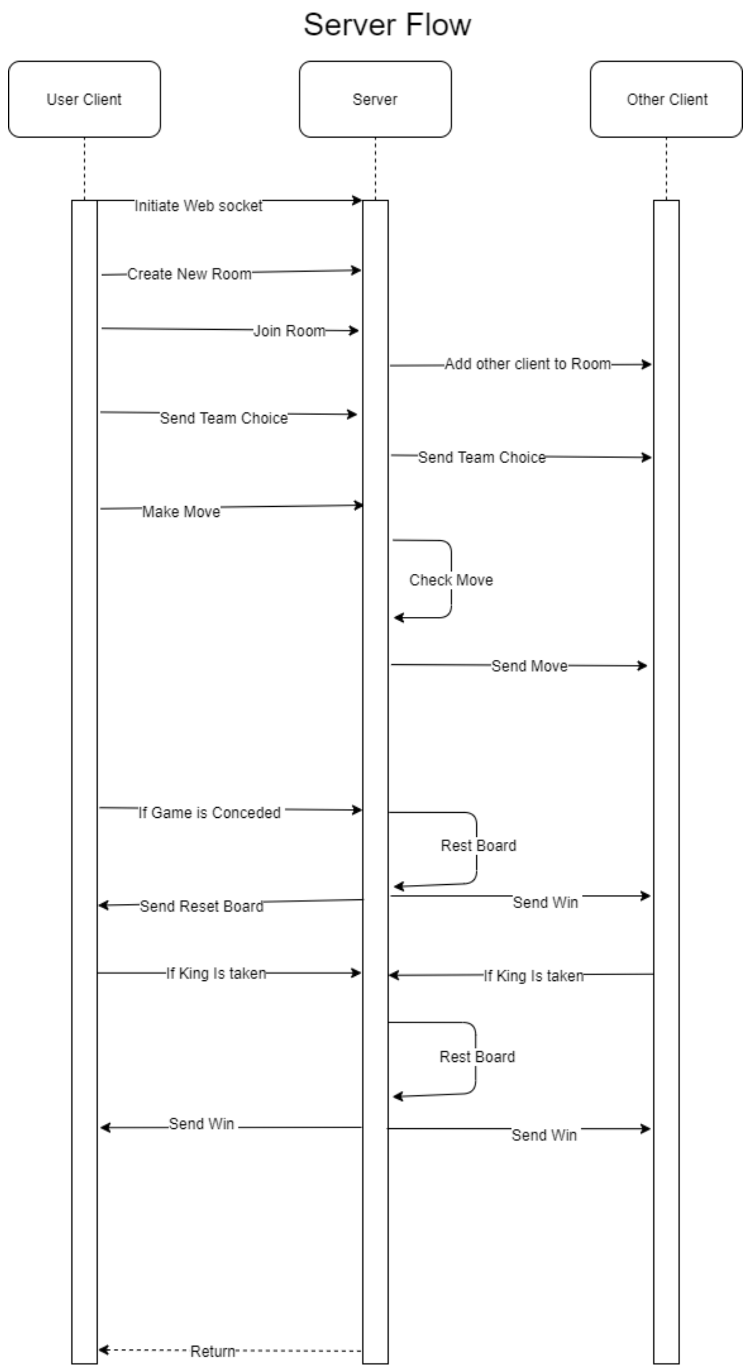
The application is aimed at people who like to play chess but don't have a chess set to hand or don't have anyone to play against locally. It is also aimed at people who want to test their skills against other people and get better at the game.

To accommodate this, I have a system for adding people to random rooms to play. This allows people to automatically be paired with another player, so they don't have to hope to find someone. The is also a system to load and save games so they can practice going through the same game in different ways. Alternatively, this can be used to load famous set ups and go from there for example Ruy Lopez set up.

The ability to set your team is also included so you can decide to play either side and not just be assigned randomly. This allows people to be play how they want and practice a side if they want to.

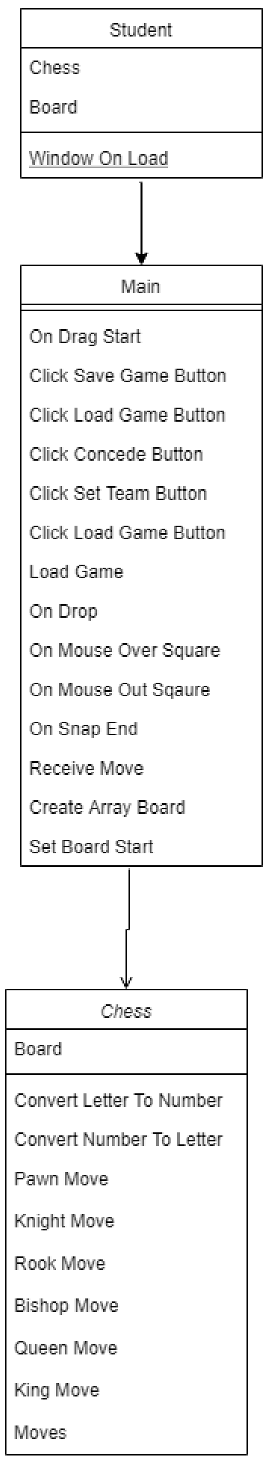
Design

The overall architecture of the project is a client that contains the logic of the game so that if the server goes down briefly the user isn't affected by the loss of connection and can make decisions. Messages that contain data are sent between the client and the server before being sent to another client. The server also handles interactions and requests from the database and the client.



The processes between the server and client use socket.io to send messages to each other and the received data determines what happens. The database is interacted with through the server creating games and retrieving them via mongoose. The logic is split in two distinct parts. The main js file where interactions from the html board object are handled and the logic of the game. This has a central access point and then is sent to the correct series of checks to determine what moves can be made. This is then passed back up to main js and the board display.

Game Logic Case Diagram



Data is streamlined for the most part. This is achieved by using FEN(Forsyth–Edwards Notation) notation to send a string to represent the board instead of an array. This comes in useful when sending a match history as strings are small than an 8 by 8 array.

This is appropriate as it allows me to make sure that the users are interacting correctly and manage the database data, so it is stored and retrieved correctly.

Testing

The main testing suites I used where mocha and chai. These where mainly used to test the logic of the game, validation and the set up. I broke my tests down into suites to make it more readable and easier to organise.

For easier comparison of arrays I wrote my own functions that check that they are the same. If they are true is return. This is then asserted by chai and the test either passes or fails.

The first suite is the utility components of the project. These include converting chess coordinates to numerical coordinates to be used in the array of the board. Its counterpart converts this numerical coordinate back in to a chess coordinate to be stored in the build. These are important things to be tested as they are used a lot and error here could cause a major issues. The other test is switching fen notation to the array board.

The testing around the pieces revolved around a pre set board state where the piece being tested was placed in the middle of the board. Around this other pieces are positioned, some on the same team others on a different team. This test the ability of the piece to move in certain directions and to take certain piece or not. This can not cover all possibilities of the positions and boards, but it can test the logic systems implemented.

For usability testing the focus was on playing the game to make it easy to play. To achieve this I had people interact with my board for the first time and see if they understood it. This was successful for the most part and people grasped the idea of dragging pieces quickly. Unlike other methods such as clicking the piece and then the square to go to. This was considered better as it gave people more time to think. This also doubled to test more of the logic system as people make random moves that would not be usually performed.

This testing strategy worked as it allowed me to test the fundamentals of my program regularly while making changes. It also revealed problems I would have missed other wise as I went along. This gave me time to fix issues when they occurred and not worry about breaking things accidently and not noticing.

Using people to interact with the board for the first time helped a to uncover other issues in the code. For example, I had messed up the logic for the knight and it was discovered by a someone testing my board. This meant I could fix it before it went further.

DevOps Pipeline

My development environment was primarily visual studio code. This is a general-purpose code editor which allows for multiple types of files to be edited. It has the ability to predict and fill in variable used in the program and highlights key words in different colours. This allows me to code easier as it highlights mistakes and reduces the amount of spelling or case mistakes that break variables.

It also has an inbuilt terminal which allows for me to launch and close the server from the editor as well as output debug information on the same page as the code. This makes hunting for bugs an error easier and quicker.

I also used Google chrome as test for the web page and its developer tools to debug the code. It allows for break points in the code and sends error logs to the counsel. This also helps a lot when it comes to debugging and testing. This also useful in seeing the experience for most users as it is the most popular web browser.

The console in the developer tools is also useful for querying the system and following the flow between the different files in the server. As well as highlighting a section of code to see the result.

The final piece of my development environment was MongoDB.com this is where my database is hosted. It has a feature to see the data currently in the database and where it is stored. This was vital for working out if the data had been sent or retrieved correctly from the server.

For my pipeline and continues deployment I used git hub. I create a repository and committed new versions to it when I completed something of note. This allowed me to have a backup of multiple instances of my project. This was useful as I made mistakes that require me to go to a previous version or take a bit code, removed from current version, from an older version.

My maim interaction was through the desktop app which allowed me to open it directly in visual studio code or folder browser.

Personal Reflection

During this project I found socket.io useful and helpful. This made communicating between the server and clients easy to implement and debug. For the most part node.js was useful as well as mongoose and mongoDB.

The biggest problem I had is using npm install to get these libraries. The default npm install can be majorly out of date leading to a lot of wasted time trying to work out why things don’t work. This made using mongoose much more difficult.

Moving forward to the next project I will learn from this. This means using @version in the npm install to make sure the version I want and think I am using is correct. This will also be much higher up in my things to check if occurs again.

On a personal note I have found this stressful and that has lead to more stress and wasting time though mistakes. I have been trying to mitigate this and will continue to try and improve in the future.

Appendix

Background Requirements

The purpose of the product is to create an online chess game which can pair people to together depending in who is online.

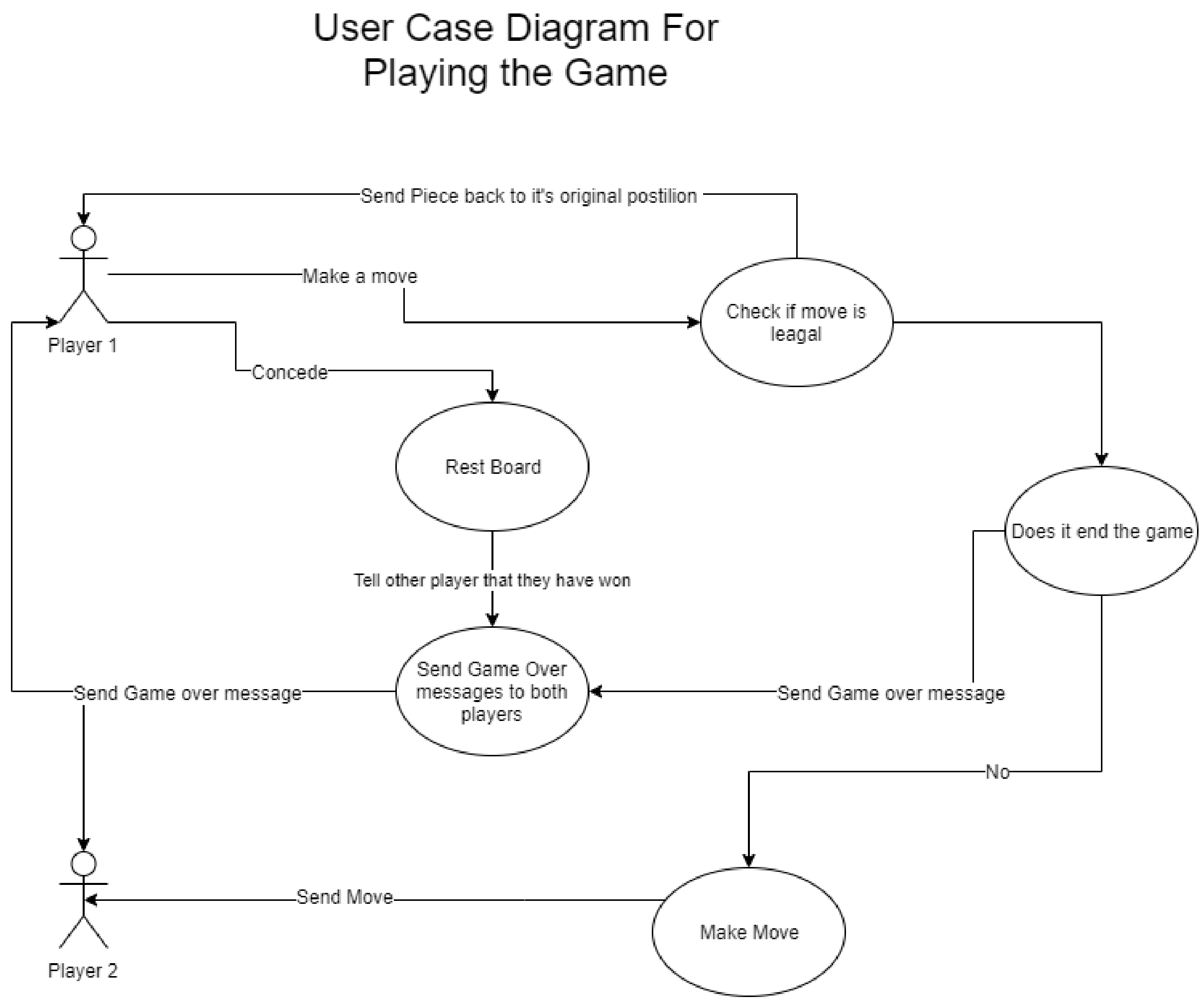
This should be an interactable board that only allows legal moves to be played. To finish the game the player either conceded or have a king taken. A game can be saved to a database by using a name the player inputs. This game can be loaded by using the same method but reading from the database. If a player leaves early the other player should win.

Requirements

- No more than two people in the room.
- The ability to send moves to the other user.
- The ability to make legal moves only.
- The ability to concede the game.
- The ability to select your team.
- Be able to save and load a game.
- An interactive board.
- The database needs to be a non sql database

Initial proposal approved.

Use Case Diagrams


User Case Diagram For Playing the Game
Send Piece back to it's original postilion
Make a move
Check if move is leagal
Player 1
Concede
Rest Board
Does it end the game
Tell other player that they have won
Send Game over message
Send Game Over messages to both players
Send Game over message
No
Send Move
Make Move
Player 2

User stories

As a user I want to be able to load games I have or some else has saved previously.

As a user I want to be able to save games to pick up later or to review.

As a user I want to be able to join a game of chess against someone else online and play a game of chess.

As a user I want to be able to concede a game early.

Aa a user I want to play the side I pick.