Contents

[Analysis 1](#_Toc122001287)

[Problem description 1](#_Toc122001288)

[Research 2](#_Toc122001289)

[Interview 7](#_Toc122001290)

[Further research 8](#_Toc122001291)

[Initial requirements 11](#_Toc122001292)

[Prototype 12](#_Toc122001293)

[Final Requirements 14](#_Toc122001294)

[Design 15](#_Toc122001295)

[Simplified full system 15](#_Toc122001296)

[Board editor 16](#_Toc122001297)

[Local Chess Game 17](#_Toc122001298)

[Data Structures 17](#_Toc122001299)

[Data Dictionary 23](#_Toc122001300)

[Pseudocode 28](#_Toc122001301)

[Validation 31](#_Toc122001302)

# Analysis

## Problem description

New technology allows us to communicate with our friends around the world, and we can also play games with them from anywhere we like. Board games have been around for centuries and they’ve always had to be played in person - sat next to each other. Toby would like to play a game of chess with his friends; although they don’t live nearby, making it hard to play with anyone. Of course, this is easily fixed by making an online version of chess.

In the case there isn’t anyone available to play with – there are still 2 different offline options in order to play chess.

One of which is puzzles. You are given a position where you must play the best move in order to continue. Typically, they are about 3 to 5 moves long and can range in difficulty from beginner to grandmaster. This puzzle feature must have a search/rating feature so that you can play puzzles which aren’t too far out of your skill level.

The other option is to create an AI, this can be created in a few ways. The easier option is to use a popular algorithm called minimax (with alpha beta pruning) – this algorithm can be used to evaluate chess positions a few moves ahead and choose what it thinks provides the most advantage. The algorithm without any optimisations is fairly slow for chess; alpha beta pruning is a way of helping this; it compares the result with the current best one and will prune it if the evaluation is lower – meaning that it stops looking down that path as it is seemingly not worth the time.

And for the online aspect, you will have the option to be randomly paired with another player or you can choose the player to go up against. This allows you to play with your friends, but also if nobody you know is online to still have a game of chess available.

## Research

In chess, there are 6 different pieces: pawn, rook, knight, bishop, queen, king (as shown respectively in Figure 1). Each move differently and have special rules in certain situations. I will use chess notation to help explain these rules and everything will be explained from white’s perspective

*Figure 1 – chess pieces*



The board

When referring to a chess board, we refer to a row as a rank, and a column as a file. For example, in the starting position, white pawns start on the second rank.

The bottom left is the coordinate A1, and the top right is H8. White’s pieces start with the ‘A’ file on the bottom left.

Chess notation

Each piece except for pawns are represented with a letter – K, N, Q, R, B. When a pawn moves, only the destination coordinate is noted down and when any other piece moves, the notation combines the destination square with the piece that has been moved. For example, Ke2 would be a king moving to the e2 square. Nf6 would be a knight moving to the f6 square. e4 would be a pawn moving to that square, with the context of which pawn being made clear in the game.

When a piece is captured, the coordinate the piece was captured on and the piece that captured it is notated with an ‘x’ in between. The pawn is once again an exception where the file the pawn was on is notated instead. Examples include: Bxe4 (bishop takes e4) or bxc5 (pawn on the ‘b’ file takes c5).

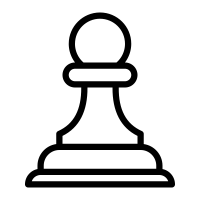
When the king is put into check, a ‘+’ is put at the end of the move. E.g. Qb5+.

Check and checkmate

When the king is attacked by an opposite coloured piece, it must move out of the attack. This can be done by either blocking the check or by moving into a square where it can’t be captured in the next move. If neither of these options is available and the king cannot be in a safe square, it is checkmate, and the opposing side wins.

The pieces

Each type of piece has their own set of moves. All pieces can’t capture their own colour, and they also can’t jump over anything – except the knight.

Pawn

The pawn is one of the most complicated pieces on the board – with more rules than any other piece. Their starting square is on the second rank.

First, when on the starting square, it has the ability to move two squares forward. Therefore, each pawn has the ability to go from X2 to X3 or X4 – where X is any available letter on the board. After a pawn has moved, they can only move one square forward.

Furthermore, when a pawn captures a piece, it does so diagonally. No matter where it is, it will always be able to capture one square diagonally forward to the left or right.

Also, pawns cannot move backwards in any way.

When a pawn reaches the 8th rank, it can promote into any piece other than a pawn or king. This is called promotion – and must be stopped at all costs in a game of chess.

There is also a special rule called ‘en passant’. This means ‘in passing’ and was a rule made to disallow pawns from escaping capture. The rule goes like this: a horizontally adjacent pawn can be captured if it has just moved two squares in the previous move. The capturing pawn will move to the square that the advancing pawn passed over as if it had move 1 square.

To describe this in context:

In this position it is black to move



d5 is played. Placing the pawns

horizontally adjacent to each other

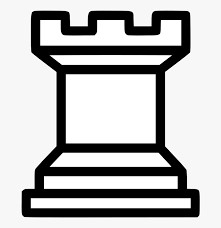


exd6 is played. Capturing the black

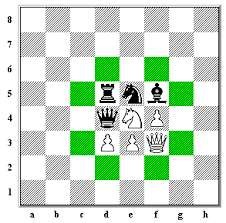
pawn.

If white was to not play exd6, it would not be available on the next move, thus en passant would no longer be available.

Rook

The rook can only move along a file or rank any number of squares that’s available. Here, the a1 rook can move anywhere from b1 to d1 and also to a2.

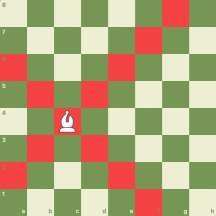
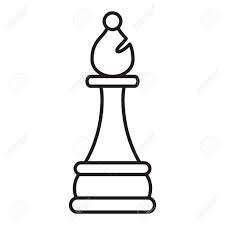
Knight

The knight moves in an ‘L’ shape in any direction, giving it 8 available moves.

As shown in figure 2, the knight can jump 2 squares forward/backward and 1 to the side; or 2 squares to the side and 1 square down/up – ignoring any pieces that are around it.

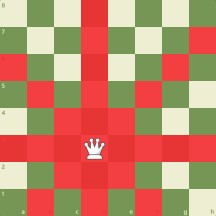
*Figure 2 - Knight moves*

Bishop

The bishop can move anywhere along a diagonal as shown in figure 3; it will never change the colour of it’s square.

*Figure 3 - Bishop moves*

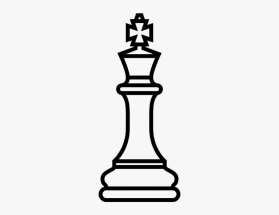
Queen

The queen has the ability to move along all diagonals and also all

files/ranks. It inherits both the movements of the bishop and the rook – making it the most powerful piece in chess.

*Figure 4 - Queen moves*

King



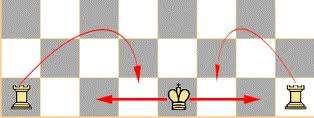
The king is the most restricted piece on the board. There are many different cases for what a king can/can’t do:

The king is capable of moving in every direction by 1 square.

The king cannot move into a square that is attacked by an opposite coloured piece; If it can be captured on the next move - it isn’t allowed.

Check and checkmate apply to the king – see earlier paragraph.

There is a move called castling – if there are no pieces in between the king or either rook, and the rook nor the king have been moved, you can move the king 2 squares across and move the rook over the king and next it.

As shown here in figure 5, since there is nothing in the way, castling is available.

However, you cannot castle if the king has to move through a check (but the rooks *can* move through the check).

*Figure 5 - Castling*

## Interview

Me: Why do you want the website to be made?

Toby: I’m passionate about the game of chess and would like to be able to play on the go.

Me: what features would you like?

Toby: Custom time-control for when I need to slot it in shorter games for when I have a free five minutes. I’d also like move takebacks and a chat feature to speak with my opponents as I like the social aspect of chess and would like that to be represented in your version. Additionally, the ability to make an account a friend’s feature would be nice

Me: what are the move takebacks for?

Toby: Well, when I make an accidental blunder or mis-click, my opponent and I can mutually agree to revert the blunder and continue a higher-level game – as opposed to losing on the spot. Similarly, time donations will also be good for this casual sort of gameplay allowing your opponent extra time as not to put too much pressure on the game.

Me: what concerns do you have for the website?

Toby: As I’m on the go, I may not have consistent network connectivity and I won’t want to lose my games permanently, so a good robust networking system would be nice. Also, I would don’t want people stealing my password or take my account.

Me: Can you elaborate on what you want for the custom time control?

Toby: I would want to be able to set a specific time and increment for each move played, and also give different time controls for each side.

Me: what’s the advantage of a high level of customisability of the time control?

Toby: it allows for players of different skill ratings to have more fair matches with each other.

### Will re-do and improve this interview…

## Further research

Making the chess website as similar to real life as possible is a big aim. So, the requirements given by my client will help me make these decisions.

Time control

In chess, there are many different ways of playing the game, namely in terms of how much time you have. Each time control completely changes the way the game is played. The 5 popular time controls are: Blitz, these typically last 3 minutes or 5 minutes; Bullet, this will last 1 minute; Classical, will last 30 minutes minimum but - in professional chess - it is 90 mins and bonus-time is added after move

30; Rapid, which is a 10-minute game; and the last most unpopular mode – UltraBullet, lasting only 30 seconds. Each player will get a timer and when it is your move, your timer counts down. When a player’s timer hits 0 seconds, the game is lost.

There is additionally an option to add extra time after each move. For example, 3-minute blitz games are popularly played with 2 seconds bonus; this means every time you make a move, 2 seconds is added to your timer. The time control in question is referred to as ‘3+2’, and a 5-minute game with no bonus will be referred to as ‘5+0’ and so on.

A different play style is required for each time control, with the lower time controls requiring quick thinking and sharpness, and the long classical games being made for long-thinkers, who are trying to find the absolute best move each turn.

PGN and FEN strings

There are a few ways of representing a game of chess. The 2 main ones are PGN (Portable game notation) and FEN (Forsyth-Edwards Notation).

PGN is more powerful than FEN, with lots of

details about a game available to you. The

main reason you would use it

is to show and

store the moves played during a game.

Whilst playing the game, this should be

created and updated automatically.

PGN is very similar to the way you would keep track of

a

real chess game.

As shown in figure 7, you would

write down the move played, along with the move it

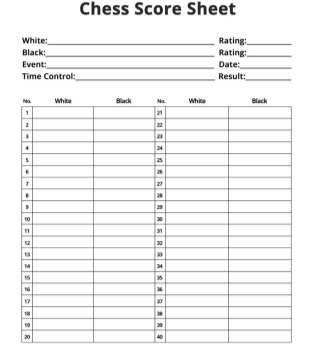
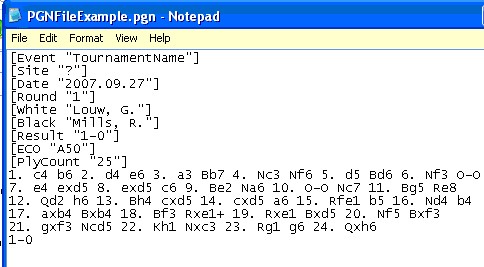
was played on. You would also write down the names

of the players and the result. PGN is essentially a

computerised version of this traditional way of keepi

ng

track of a chess game.



*Figure*

*6*

*-*

*PGN example*

FEN strings have a completely different use. Rather than describing a full game of chess, it describes one position – allowing you to pick up from where you left off at any moment. It tells you about castling rights and also how many moves into the game you are.

An example fen string would look like**:**

**r1b1k2r/p3bppp/1pnqpn2/8/3P4/P1NBBN2/1P3PPP/R2QK2R w KQkq - 0 11**



The string above represents this board. The characters before the first slash (r1b1k2r) represent a8 to h8; the next set (p3bppp) represent a7 to h7 and so on. Lowercase letters are black pieces and uppercase letters are white pieces. The numbers tell you the squares are empty.

At the end of the string there is ‘w KQkq – 0 11’. This isn’t describing what’s on the board but instead, tells you some information about the board. The ‘w’ means it is white to move, where it would be ‘b’ if its black’s turn. The KQkq is describing the castling rights available. ‘KQ’ means that white can castle Kingside and Queenside and vice versa. If these rights aren’t available, the letter simply won’t show.

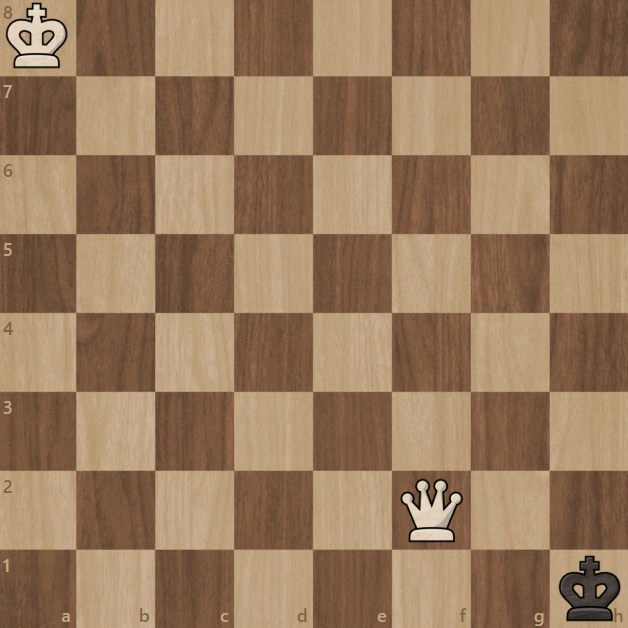
If a pawn has moved 2 squares, the ‘-‘ will say the coordinate the pawn passed over. For example, in the starting position, if e4 was played, the FEN string would be:

‘rnbqkbnr/pppp1ppp/4P3/8/PPPPPPPP/RNBQKBNR b KQkq e3 0 1. This is to indicate if en passant is available.

The second last number indicates how many half moves have been made since the last pawn capture or advance. This allows you to enforce the 50-move rule, where if neither player has pushed or captured a pawn in 50 moves, the game will end in a draw. Therefore, when the counter reaches 100, this indicates that 50 full moves have been made since the last pawn move/capture.

Finally, the last number shows how many full moves have been made in the game as a whole. Therefore, it is incremented every time black moves.

Uncommon Game Ending Conditions

There are a few conditions which end the game, not including checkmate.

Stalemate is one of these, if it is a player’s turn and there are zero legal moves they can play, and the king isn’t in check: it is a stalemate and the game ends in a draw. The attacking side needs to avoid this as it will typically only happen in case where they have a big game winning advantage.

Here in this position, it is black to move, but there are no moves to play, and the king isn’t in check, thus it is a draw.

Another condition is the 50 move rule, if a pawn hasn’t been moved or a piece hasn’t been captured, the game ends in a draw. This will typically happen in an endgame where there a very few pieces on the board.

Next, there is the draw by repetition, if the same position occurs 3 times, then the game can end in a draw.

Finally, there is a draw by lack of sufficient material, this means that if both sides don’t have enough pieces to make a checkmate, the game must end in a draw automatically. For example, a knight vs king cannot make a checkmate by itself (so the game ends), but a knight against a king and any piece is possible (due to a rare position in which a player may trap their own king). Also, if it is a king and pawn alone, despite the fact a single pawn cannot checkmate, the game can continue as the pawn can promote.

## Initial requirements

1. Display a game of chess.
2. Allow the user to enter in custom positions through a FEN string.
3. All pieces move legally
4. Have alternating moves
5. Show the legal squares that pieces can move to.
6. If it is white’s move, don’t display legal moves if you try to move a black piece.
7. Game ending conditions.
8. Pages for Home, online matches, board editor, local game and puzzles.
9. Allow users to connect to game through home page
10. Allow users to create custom games from home page
11. Display list of users currently searching for a match.
12. Display black and white’s respective pieces on the client’s side.
13. Display timers in game for both colours.
14. Be able to join the same room after leaving.

## Prototype

Using the initial requirements, I came up with a front-end interactive website. It works for any browser on a laptop or PC (on any operating system) and is dynamically sized; allowing for any screen size to open up and play chess.

Graphical user interface, chart, treemap chart

Description automatically generated

First thoughts:

There’s a too much unused space on the page, could do with either moving the boxes around, or adding more features.

The background is too plain and simple, I’d like it to pop out more, seems monotone at the moment.

Can you set your username anywhere? It would be nice to see who I’m playing against and also set my own name too.

The header navigates you to each page. The buttons highlight when hovered and are the same for each page, keeping consistency across the website.

Good choices for time controls chosen and the format is nice. Custom is a good bonus too.

The most common game types are given, and if the user wants to have complete freedom over their time control, the option is there.

Font-size is nice and readable, but each option could use some grid/boxes to separate them out.

Chart, treemap chart

Description automatically generated

When players join a game, their ID and time control chosen is displayed here. This includes custom options chosen.

This could use with some more intuitive formatting. Perhaps make the colour of yourself different. Is it possible to join this game by clicking on it?



This suggestion is a nice improvement to the card. I’ll give the user the ability to click on the text in the PGN and go back to the position when that move was made. Or add an arrow to navigate the positions.

I like the idea; it would be nice if you could look at previous moves as well.

This card will be used to display the game information. It will display moves made (in the PGN format).

First thoughts:

The board looks good, I think that the colours are easy on the eyes and the piece set is smart.

I like the position of the side card. It doesn’t take away from the board itself and seems like it is easy to quickly glance at.

Will there be coordinates on the board? I use them frequently on a real board and beginners typically like to use them.

This board and card combination will be used for both local play and online play. The colours of the board can be customisable, and you can also change the piece set.

Graphical user interface, application

Description automatically generated

Here is the FEN string for the current position on the board. It is updated live as you place pieces. It also scrolls if the text goes over the edge

First thoughts:

I like how the pieces to place are positioned intuitively; I feel I know how to use the tool straight away.

The scroll bar underneath the FEN string isn’t obvious at first, maybe hide it until it’s needed?

I don’t like the position of the FEN string; I’d prefer if it was underneath the castling box as they are related.

The white to move box is too small and the placement is odd. Perhaps move the text to the bottom.

I think that a button to copy the FEN string would be nice, especially since you have to scroll quite far when the string is long.

This is the board editor page; you must choose a piece by clicking it and then click/drag to place it.

This can be solved by adding a ‘mouse’ option on the piece-selection bars.

The piece placement is good, but I’d like to be able to pick up pieces instead of having to re-select one each time.

## Final Requirements

Using my initial requirements and research. I have finalised what is necessary to make a website my client wants.

1. Display a game of chess – this should be fully customisable, with your own board colours available and piece sets, along with beginner friendly options like coordinates and showing the legal moves a piece can make.
2. Allow the user to enter in custom positions through a FEN string. A board editor page will allow the user to place pieces where they’d like; the pieces positions will then be calculated, and a FEN string will be outputted. There will also be the option to open a new local game page with the custom position the user creates through an easy access button on the page.
3. All pieces must move legally; as found in my research, there are many edge cases which must be accounted for, this includes en passent, castling through check; or pinned pieces – just to name a few.
4. Have alternating moves – the system should automatically disallow white/black from playing 2 moves in a row, as per the rules of the game.
5. Show the legal squares that pieces can go to. This may be optional for the user, and will help beginners see where pieces can go, instead of having to memorise the legal moves straight away. A more experienced player may want this to be turned off.
6. If it’s white’s move, don’t display the legal squares if you try to move an opposite-coloured piece in order to avoid confusion with what is actually a playable move.
7. Game ending conditions: through my research, I found there are a few ways a game can end. Each of these situations must be programmed and automatically decided for the players.
8. Home, online matches, local play, board editor and puzzles pages. Each of these are the main functions that my client would like to use - as per the requirements specified in the interview.
9. Allow users to connect to game through the home page: a matchmaking system must be in place where users can choose a time control and be placed in a match with other players accordingly. The prototype shows how this may work and look.
10. Custom games from the home page must also be available, where the user can pick any time control and increment they want, and other users can see and join this game.
11. Display list of users currently searching for a match, this can be a short list where the user can choose to see what types of game is displayed (show all users searching for 1+0, or 3+2 etc.).
12. Display black and white’s pieces on their respective sides – when the clients are connected to the server, they are of course both different colours, this means displaying their colour on their side individually.
13. Display timers for both colours – this should be obvious which side the timer is for.
14. Be able to join the same room after leaving. If a client disconnects either by choice or due to an issue, they should be able to get back into the game without any issue.

# Design

## Simplified full system

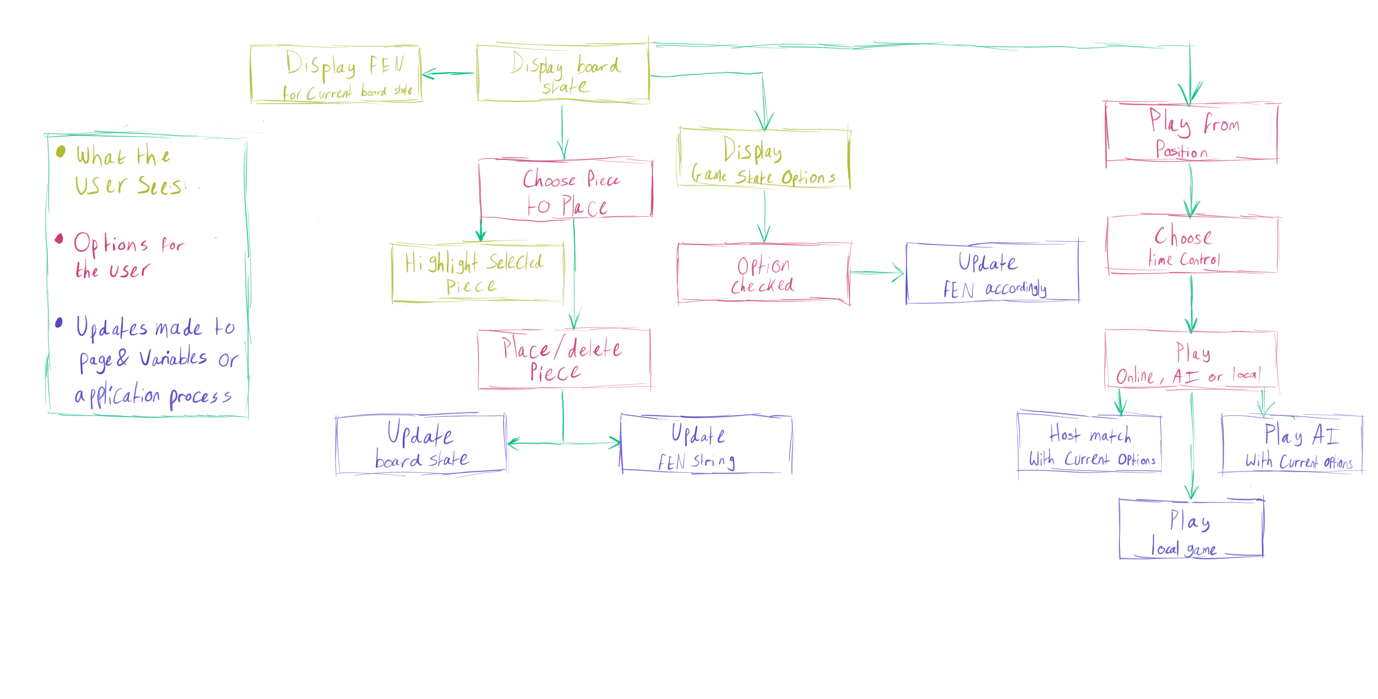
Diagram

Description automatically generated

This diagram shows how the main navigation of the website works. The colours are explained in the diagram.

When the user first connects, they join the waiting room, and the home page is shown; this is where they can select an online match to search for and also see other users who are currently searching for a game. When an option is selected, it will check all the other users who are currently searching for a game, and if somebody is searching for the same time control, the 2 players will be matched and put into a game room together.

## Board editor



The board editor is where the user can create their own position. In order to place pieces, you must select them from the menu and click anywhere on the board. You can also choose castling rights, and whether or not it is white’s move. It outputs a FEN string for the position you created which you can copy and paste to save the position for later. Alternatively, you can host either a local game; online match; or play the AI; straight from the page with a button which will redirect you to the one you choose.

Once the host option has been chosen, the user must enter a time control and then pick one of three options:

* If the online match option is picked, you will be given a room code which can be shared with someone so they can join you. The room code will be searchable in the custom option on the starting home page.
* If the local match is picked, the user will be redirected to the ‘pass and play’ page where the custom position will be in place.
* If the AI option is picked, the user will be able to pick the level of the AI, and will then be redirected to a board with the bot.

## Local Chess Game

Diagram

Description automatically generated

This diagram represents how the game works for local play. This diagram is in fact very similar to online play; however, online play doesn’t flip the board after a legal move has been played. This is because each player should already be on their respective sides, so the ‘pass and play’ aspect is no longer there. Thus, everything else about online play is the same.

When a piece is clicked, it shows the squares which it can move to; this may be a distraction to more experienced players and will be optional, with the ability to turn it off in a preference menu.

Simultaneously, the piece is drawn on the cursor, giving intuitive feedback on which piece you are currently holding and where it will be placed.

When the game is won (through any of the ways discovered in my research) – a screen will display saying the winning colour, with the option to play another game. If both users agree to a rematch, the match will restart in the same room, with the colours flipped in order to make it fair.

There will also be the option to see any of the previous moves. If the player clicks on a move through the PGN system, it will show the position in that moment. This can be useful for analysis on your own game, or if you weren’t looking at the screen in the moment they played a move.

## Data Structures

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Property Name | Description | Data Type | Example |
| Board | occSquares | A 2D matrix which stores all pieces on the board | Piece | [0,0,Piece,0,…] |
|  | moveCounter | Stores the number of moves played in the game. It increments each time black plays a move | Integer | 3 |
|  | whiteToMove | Indicates whether it is black or white’s turn | Boolean | true |
|  | blackShortCastlingRights | Indicates if a side can castle long or short. each of these castling right Booleans are set to false when a respective castle-disabling move is played. | Boolean | true |
|  | blackLongCastlingRights | See blackShortCastlingRights | Boolean | true |
|  | whiteShortCastlingRights | See blackShortCastlingRights | Boolean | false |
|  | whiteLongCastlingRights | See blackShortCastlingRights | Boolean | false |
|  | maskMap | A 2D matrix where each time a move is made, you must check if your king is under attack. Maskmap is used to mask occSquares with squares that are attacked by the opposite side, thus checking if the move you played just put your own king into check. | Integer[] | [1,0,0,1,0,0,0,1  0,1,1,0…] |
|  | pawnMovedTwoSquares | If a pawn on a starting square moves two squares, this must be stored in order to check for available en passent moves. | Boolean | true |
|  | pawnMovedTwoSquaresCol | The column which the pawn moved 2 squares is stores, thus allowing you to capture the pawn on the respective column. | integer | 6 |
|  | enPassentTaken | If en passent is taken, it is a special case move which requires its own function. Thus, when this variable is true, the program will call the en passent function | Boolean | false |
|  | isInCheck | In order to check if the game has been won, the game needs to know if the king is in check, and also has no moves. | Boolean | true |
|  | shortCastles | Short castles and long castles are more special cases which require their own subroutine. These variables are set to true when the player does the respective option | Boolean | true |
|  | longCastles | See shortCastles | Boolean | false |
| Piece | type | Each piece type is assigned a number. This allows or easy storage and access (more on *colour)* | integer | 1 = king  2 = pawn  3 = knight  Etc. |
|  | row | Stores the row of the piece | integer | 4 |
|  | col | Stores the column of the piece | integer | 3 |
|  | colour | The colour is stored as 8 for white and 16 for black.  This is so that a piece can be accessed as a 5-bit integer and allows for efficient bitwise operations.  e.g. white pawn = (white XOR pawn) = 10 | Integer | 8 = white  16 = black |
| PieceType | type | A data dictionary to convert a piece letter to a number | integer | {‘k’: 1, ‘p’: 2,…  ‘K’: 1, ‘P’: 2} |
|  | numToType | A data dictionary to convert a coloured piece to a type | char | {9: ‘K’, 13: ‘R’…,  18: ‘p’, 21: ‘r’} |
|  | numToPieceName | A data dictionary to convert a piece type to the full name of piece | string | {1: ‘king’, 4: ‘bishop’} |
|  | none | All of these below store the number to indicate the piece type | integer | 0 |
|  | king |  | Integer | 1 |
|  | pawn |  | Integer | 2 |
|  | knight |  | Integer | 3 |
|  | bishop |  | Integer | 4 |
|  | Rook |  | Integer | 5 |
|  | queen |  | Integer | 6 |
|  | white |  | Integer | 8 |
|  | black | See *none* above. | integer | 16 |
| Front | p5 | As this program will be using ES6 modules, it must pass in the instance of the canvas library I am using called *p5* into the *Front* class | p5.js | p5.createCanvas(400,400) |
|  | black | Decides what colour the black squares are on the board | string | Rgb(0,0,0) |
|  | white | Decides colour for white squares on the board | string | Rgb(255,255,255) |
|  | spacing | When the piece is either picked up or drawn on a square, spacing makes sure it is centered on the mouse or in the square | float | 364.5234 |
|  | blocksize | This is the size of the squares in the grid, it is calculated by the smaller value of length and width divided by 8. This is to ensure it stays as a square on an 8x8 grid. | float | 32.486 |
|  | pieceScale | Is how big the piece is when drawn. | float | 0.85 |
|  | images | An array of p5 image objects which stores the png’s for all the pieces. | p5.image | Front.image[4] |
| Timer | clientIsWhite | The timers must be drawn on opposite sides for each client.  For example, white wants to see their timer on the bottom (their side) and black also wants to see their timer on the bottom (their side). | Boolean | true |
|  | time | Time is stored in seconds. | integer | 180 |
|  | increment | Each time a player makes a move, a small amount of extra time is added, this increment is decided when the player searches for a match | integer | 2 |
|  | isWhiteTimer | Is used to indicate if the timer is for black or white. | Boolean | true |
|  | timeDisplayed | Is displayed in the format (MM:SS). A function converts the seconds left into this format.  Each time a move is made:   * Log the time in which the move was made * Subtract this from the time in which the move started.   This calculates the time taken to make the move, thus allowing *time* to be updated. | String | 01:58  11:23  00:32 |
|  | tempTimeToDisplay | Is shown only to the clients and decrements as close to 1000ms each second. May be offset by a few ms, but the drift should constantly correct itself. | float | 1001.9912  998.5422  1000.0023  1005.1543 |
| Game  Rooms | roomCode | Stores the randomly generated roomCode | string | Ar]Ffl |
|  | board | Stores a 2D matrix of the board class on the server, allowing it to be sent to anyone who joins | Board | board.occSquares |
|  | whiteTimer | Stores how much time white has left in the format ‘MM:SS’ | float | 00:34 |
|  | blackTimer | Stores how much time black has left to move in the format ‘MM:SS’ | float | 02:21 |
|  | PGN | As the game is played, the PGN is updated so that players can see the previous moves they have played. This can then be outputted to the players if they’d like to download it. | string | 1. e4 e5 2. Nf3 nf5 |
|  | client | Stores the socket ID of the client, this can be used to send them any information over the websocket regarding the board or player messages. | string | 6n\_k\_L4iicOkuWIaAAA |

## Data Dictionary

**Online/local chess game**

|  |  |  |  |
| --- | --- | --- | --- |
| Variable name | Description | Data type | Example |
| canv | Stores the p5js Canvas | P5 | Canv = p5.createCanvas(500,500) |
| canvasDiv | This is how the p5 canvas communicates with the HTML DOM | ? | Document.getElementByID(‘….’) |
| board | Stores the instance of the Board class in which the game is played. | Board | Board.isLegal(…); |
| front | Stores the front-end variables like colours and spacing | Front | Front.spacing |
| BLOCK\_SIZE | BLOCK\_SIZE  ->  BIN\_PIECES  Used to initialise properties in front class | float | 45.723 |
| PIECE\_SCALE | ^ | float | 0.85 |
| WIDTH | ^ | float | 567 |
| HEIGHT | ^ | float | 800 |
| SPACING | ^ | float | 15 |
| IMAGES | ^  Stored in public/classic\_hq | p5 image object | ‘b\_bishop.png’  ‘w\_pawn.png' |
| BIN\_PIECES | ^  Data dictionary which converts piece and colour types to names used to read in png images | string | 20: ‘b\_bishop’, 17: ‘b\_king’ |
| size | Smallest value out of width and height | float | 753.5678 |
| bitmap | 2D matrix used to check for attacks against the king | integer | [0,1,9,10…] |
| legalCircles | When a piece is clicked, a 2D matrix is created, showing the legal squares available for a piece to move to. | string | [‘e5’, ’f6’, ’g7’] |
| mouseDown | Set to true when the mouse is being clicked | Boolean | true |
| pieceAtMouse | When a piece is clicked, the piece object is returned, providing all methods and properties of the piece | Piece | Piece.type == 3 |
| selectedCoords | When the canvas is clicked, the clicked coordinates are returned as a 2D matrix, providing the row and column. Is used for playing moves. | integer | [2,3]  [0,1]  [0,7]  [7,7] |

Online uses all of the above plus more:

|  |  |  |  |
| --- | --- | --- | --- |
| Variable Name | Description | Data Type | Example |
| socket | Is how the web server and client communicate. Uses the socket.io library. | Socket | Examples at Documentation:  <https://socket.io/docs/v4/> |
| queryString | Returns a queryString part of the URL which is a string preceded by a ‘?’ and then can be later followed by ‘&’ | string | ‘?time=1&increment=0’ |
| urlParameters | Is a URLSearchParams object where it is used to access the parameters. These parameters can then be assigned to variables using by querying them | URLSearchParams | urlParameters.get(‘increment’) |
| time | This is the time control chosen by the players when the searched for a match | integer | 3, 5, 10, 1 etc. |
| increment | Is the Increment chosen by the players when they searched for a match | integer | 1,2,3,5 etc. |
| clientIsWhite | Is the colour that they will play in the game; it is randomly assigned to the player when they join the lobby | Boolean | 0, 1 |
| roomCode | Is a 6 characters string of random characters, ranging from ascii values of 48->122, giving 6^74 different possible codes | string | Ar]Ffl |
| blackTime | Stores the Timer class for black | Timer | blackTime.time = 43 |
| whiteTime | Stores the Timer class for white | Timer | whiteTime.increment = 2 |
| timeMoveStart | When a move is made, the time it was played it stored, this can later be used to calculate the time taken, which will be done through the server in one calculation.  It uses the javaScript Date.now() function which returns the time elapsed in milliseconds since 1970. | float | 543.23421 |
| timeMoveEnd | Is |  |  |
|  |  |  |  |

**Server-Side chess game**

The server also stores an instance of the chess game, along with move validation. So, all variables above are also stored on the server along with the extras below.

|  |  |  |  |
| --- | --- | --- | --- |
| Variable names | Description | Data Type | Example |
| matchmaking | Stores all sockets currently searching for a game in a dynamic 2D array. When players search, the matchmaking will pair players who searched for the same time control and increment. Once the players have found a game, they are spliced from the queue, so that others don’t also match with them. | integer | [3,2] -> [time,increment]  [5,0] |
| GameRooms | Stores all sockets currently in a game who have found a game through the matchmaking system and the rooms can be accessed through the room randomly assigned room codes, as opposed to an index.  Is stored as a GameRoom class as stated earlier. | GameRoom | GameRoom[roomCode].board |
| alreadySearching | If a player is already in the matchmaking queue, this will be true.  It can be used to: rather than search for a new match, change the current match they are searching for. This stops there from being duplicate users on the matchmaking list as the users preferred game gets updated instead of pushed. | Boolean | true |
| gameFound | If 2 players are searching for the same time control and increment, gameFound is set to true and the players are paired and queued up into a match together. | Boolean | true |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Board editor**

|  |  |  |  |
| --- | --- | --- | --- |
| Variable Names | Description | Data Type | Example |
| selectedPiece | When a piece is clicked on the selection menu, it will be placed anytime you click on the board.  In this mode, if you click on a square that isn’t empty, it will overwrite that square with your currently selected piece. | Piece | Piece.colourAndPiece() |
| BoardFEN + endOfFen | The board editor’s main function is outputting a FEN string which the player can then use. So, on each new piece placed/moved/deleted – the FEN string will be updated accordingly | string | 8/8/8/8/8/8/8/8 w KQkq - 0 1 |

## Pseudocode

### Find illegal King squares

**First level pseudocode:**

if black to move then find attacks from white

else reverse

iterate over board

if iterator is on a piece you want attacks from

find the type

find all attacks from that piece type

if it’s an empty square set it to 1 or if it’s the king then set it to 1

if it’s the same-coloured piece then leave it

return created bitmap

**Second level pseudocode:**

colourCalc <- 16

if findAttacksFromWhite == true then colourCalc <- 8

for i <- 0 to 7

for j <- 0 to 7

if (position[i][j] <> 0 AND ((position[i][j].colour & colourCalc) = colourCalc))):

switch (position[i][j].type)

case type is Knight, King, Pawn:

for (options of position[i][j].intervals)

col <- j + options.dx

row <- i + options.dy

if (isOnBoard(row,col)):

if (position[row][col] = 0):

bitmap = 1

else

if(position[row][col].type = King and position[row][col].colour & colourCalc = 0):

bitmap[row][col] = 1

default:

for (options of position[i][j].intervals())

col <- j + options.dx

row <- i + options.dy

while(isOnBoard(row,col)) //while hasn't gone outside of board

if (position[row][col] = 0) then

bitmap[row][col] <- 1

else //if a piece has been hit

if (position[row][col].type = PieceType.king and (position[row][col].colour & colourCalc) = 0) then //if its the king then continue

bitmap[row][col] <- 1

else

if (position[row][col].type !== PieceType.king) then //doesn't store the same coloured king as a valid piece to take

bitmap[row][col] <- 1

break //break out of while loop

### Connect to online match

**First level pseudocode**

If already searching then update the match type

Else push player to matchmaking queue

Iterate over current matchmaking list

If another player is searching for same game type

Generate a room code

Randomly generate colours for players

Redirect players to game room

Check if the game room exists

If not, instantiate game room class onto server and push into Game Rooms

Array

Remove players from matchmaking queue

**Second level pseudocode**

for (k <- 0 to LEN(mm)

if (mm[k].id = data.id //data is the time control and increment sent from home

mm[k] = data

alreadySearching = true

if (NOT alreadySearching) mm.push(data)

for i <- 0 to LEN(mm)

for j <- i + 1 to LEN(mm)

if (mm[i].time = mm[j].time AND mm[i].interval = mm[j].interval)

Clients = sockets.get(‘WaitingRoom’)

roomCode = generateRoomCode()

colour1 = getRandomInt(0,1)

colour2 = absolute(colour1 – 1)

for (clientID of clients)

clientSocket = sockets.get(ClientID)

if (ClientID = mm[i].id)

data = {

client:mm[i].id,page: ‘/onlineGame’, ‘roomCode’: roomCode, isWhite:colour1

}

clientSocket.emit(‘redirect’, data)

Else if (ClientID = mm[j].id)

Data = {

client:mm[j].id, page:‘/onlineGame’, ‘roomCode’: roomCode,isWhite:colour2

}

clientSocket.emit(‘redirect’, data)

if (not gameRooms[roomCode] exists)

gameRooms[roomCode] = new GameRoom(roomCode, board, whiteTimer, blackTimer, PNG, [])

matchmaking.splice(j)

matchmaking.splice(i)

### Check if the king can castle

**First level pseudocode**

Function checkKingRank(king, dir)

Iterate of direction you want to castle in

If the iterator hits a square that isn’t empty

If this square is attacked by an opposite piece

Can’t castle through the check so return false

If the piece hit is the same colour rook on the *h* square

Is a valid piece to castle with so return true

If the piece hit is the same colour rook on the *a* square

Is a valid piece to castle with so return true

Else the piece hit was not available to castle with so return false

Return false

End function

**Second level pseudocode**

Function checkKingRank(king,dir)

for (i <- dir; absolute(i) To 4 step dir)

if (this.maskMap[king.row][king.col + i] <> 0) then //if piece has been hit

//if square found is attacked by opposite colour

if (Math.abs(i) < 2 && this.maskMap[king.row][king.col + i] = 1) then

return false

//if piece is same colour rook on the 'h' square

if ((king.col + i = 7) && (this.maskMap[king.row][king.col + i] <> 0)) then

return true

//if piece is same colour rook on 'a' square

else if((king.col + i = 0) && this.maskMap[king.row][king.col + i] <> 0) return true;

else return false

return false

End function

## Validation

**p5 canvas library**

p5 is the library used to interact with and display the board. It has its own mouse events and own coordinates relative to the board (top left being (0,0)); this sometimes conflicts with the html/jQuery mouse events. If you click outside it, the script will crash, and the board will become unresponsive. In any page where the board is displayed, a check must be made to ensure you are never going outside the board itself.

The pseudocode for such a function may look like this:

Function isOnBoard(Row,Col)

if (Row >= 0 AND Row < 8 AND Col >= 0 AND Col < 8) THEN

return true

return false

End function

The p5 mouse event returns the number of pixels the cursor is relative to (0,0). Thus a (mouseX,mouseY) may look like (640,320). The *isOnBoard* function will be able to