

Contents

1 Analysis	4
Background to the Project	4
Current System	4
Prospective Users	4
Interview	5
Diagrams of existing systems	7
User Requirements	8
Objectives	8
Technical Research	9
Problem Modelling	10
Hardware and Software Requirements	10
2 Design	11
High Level Overview	11
Database Design	14
Entity Relationship Diagram	14
Overview of SQL Queries Used	14
Normalisation	15
Cascade	15
Data Flow Diagrams	16
Level 0	16
Level 1	16
Level 2 (Focused on the Backend)	16
Data Structures	17
User Interface	19
Prototype Screen Designs	19
General Design and Accessibility	23
Algorithms	24
Elliptic Curve Signing Algorithm	24
Alpha-Beta Pruning	28
Board Hashing	29
Adding Positional Weaknesses	30
Move Generation Algorithm	31
Piece Counting Algorithm	32
Hardware and Security	33
3 Testing	34
Test Plan	34

Test 1 - User interface for the board	36
Test 2 - Attack Generation	38
Test 3 - Move Generation	40
Test 4 - Engine	43
Test 5 - Cryptographic functions	44
Test 6 - Login and Signup form	45
Test 7 - API and Authorisation	47
Test 8 - Custom Game and Review	48
Test 9 - Adventure Mode	49
Test 10 - End-to-End testing	50
Screenshots	53
Testing video links	57
Transcriptions	57
4 Evaluation	59
Changes Due to Feedback	59
Project Objective Evaluation	59
Possible Improvements	60
5 Code	62
File Hierarchy Diagram	62
File Descriptions	64
Cover Sheet	67
Appendix	69

1 Analysis

Background to the Project

With the recent boom in popularity of chess in the last couple of years, there has been an increase of demand for tools to help people improve in the game. However, many of these tools are computationally expensive, and therefore are locked behind paywalls and subscriptions such as chess.com's game analysis system. My NEA will be a clear-purpose web application that provides an analysis of the user's weaknesses and adapts the computer playing against the user to help the user improve, overcoming the issues of expensive computation by analysing positions in the user's own browser.

Current System

There are many options available to people trying to improve at chess, outside of just playing against people. Automated tools include:

- Revision and memory tools to help the person learn checkmating patterns and opening lines.
- Puzzles to improve tactic spotting.

Those two will be out of scope of this project, what I will looking at to focus on and improve are:

- Playing against engines.
- Analysis of past games.

Most chess websites provide systems to play against engines at different strength and some provide the ability to change between different 'personalities' which will then change the playing style of the engine. However, many of these 'personalities' are not free, and there is a limit of how much customisation can be done. For my project, I don't want to make a super-powerful engine, but rather focus on a high level of customisation which is much more important for people learning chess below the average skill level. To get this high level of customisation I will have to forfeit some of the speed and therefore depth in the engine.

Secondly, instead of providing a game-by-game analysis of each game, I instead want to analyse the entire history of games played by the user on my application. This should give more insight to the playing style and positional errors made by the user in their chess games, as a game-by-game analysis causes the user to focus on tactics they are missing instead of the big picture.

These changes make my project not a replacement for existing chess analysis programs, but something than can be used alongside traditional analysis programs.

A major problem suffered by chess analysis programs are the price of analysing games. Chess engines require a lot of processor time on the server, which is expensive, explaining the restrictions implemented on the number of games that can be analysed on many websites. To get around this, and provide more independence for the client, the engine will be run in the web browser.

Prospective Users

This project is designed for people who know at least the basics of chess, and hopefully have played some chess online, so the website will feel familiar and not overwhelming, and the user should

understand basic terms of chess analysis. The engine will not be suitable for advanced chess players as it the engine won't have a depth of analysis high enough to beat someone who has played chess for a long time (due to the computational limits of browser-based analysis).

For people who have just learnt chess, a playstyle analysis program will not be as helpful, as newer chess players tend to have more sporadic playstyles which have less use analysing.

The targeted audience for this website is younger people (around 12-17 years old) who have picked up chess in the recent boom of chess popularity online in the last couple of years and are struggling with positional issues in their chess games. I will use a group of people at my school's chess club to evaluate this project at range of skill levels.

The website will only be able to run on desktop computers, as mobile phones are less likely to have the hardware and modern browser features to allow analysis in the browser.

All users will interact with the website as clients and will play games on the site against the engine, no matter how advanced they are. The user will be able to adjust the engine manually.

As the project might be used by younger users, the design of the website must be simple and intuitive to use and should be approachable by people who have never played chess online. The application should also meet accessibilities requirements for a website, by using semantic HTML elements and ARIA tags when that is not available.

The project should be designed in a way so that it can be run by a client who doesn't have too much experience with computer science. Possibly clients include schools, a group of friends, or a parent, who could run their own instance of the web server, providing the system to students, other friends, or children.

Interview

Interview with a friend who is very good at chess. The interviewee also has experience working with younger children, as he tutors younger students weekly, so he made a perfect candidate to have a technical discussion about what the system could provide my users.

Me: When younger children are using a learning tool for a game such as chess, what kind of features will keep them engaged specifically for a game like chess?

Answer: Definitely the most important [feature] is a sense of competition. [The users] wanna show off to their friends with a wide range of metrics. It'll also be cool to see some sort of horde mode, or game with some kinda objective.

Me: Any other ideas?

Answer: [The existing systems] aren't user friendly, and most kids, you know, get confused from messy apps.

Me: With the actual bot, what could I provide which existing computers don't?

Answer: [The engine] could, like, be really, really aggressive, and just start a massive attack on your pieces or be really passive and just keep its pieces close, cause even, like, chess.com doesn't have that much really customisation, they all kind of feel the same.

After the preliminary interview, I created this table of potential ideas to give to the interviewee in a secondary interview.

Me: Rate these features on a scale of how often you would use them. (I present the interviewee with a table with headings Never, Rarely, Sometimes, Frequently, Always, for him to tick)

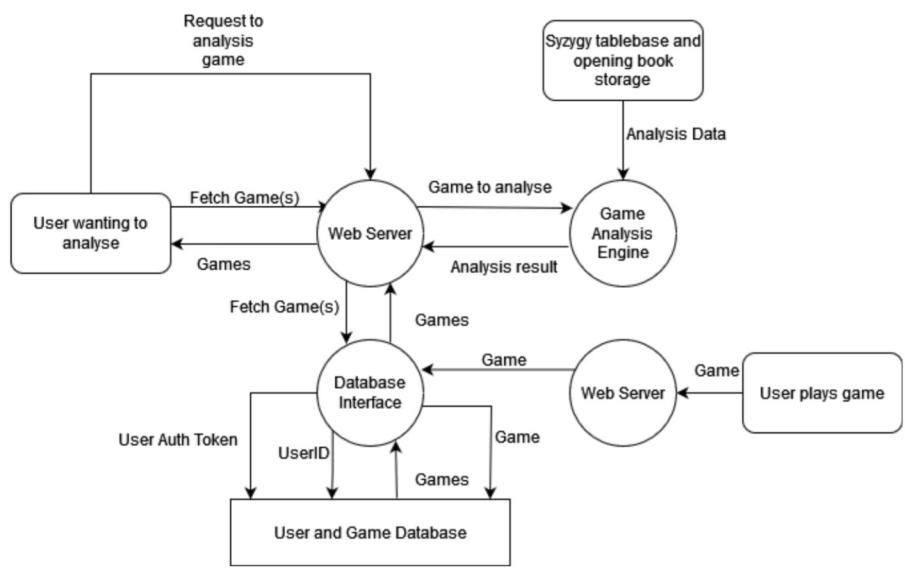
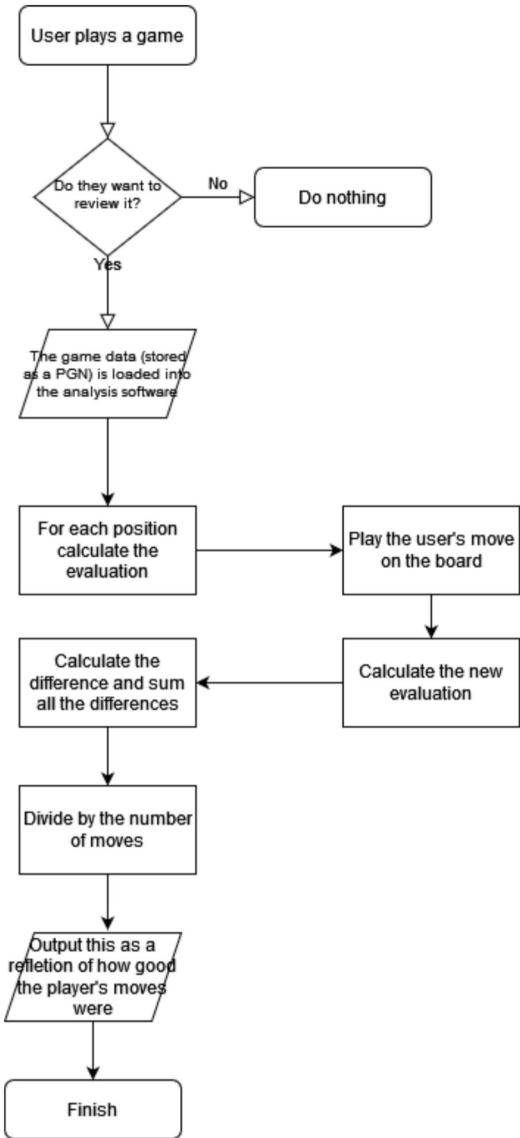
	Never	Rarely	Sometimes	Frequently	Always
Lifetime Playstyle Analysis					(Ticked)
Survival Mode with endless levels		(Ticked)			
Adaptive Engine*	(Ticked)				
Adventure story-based mode				(Ticked)	
Sharing your lifetime playstyle with friends					(Ticked)
Manual adaptation of engine playstyles					(Ticked)
Predetermined engine settings to play against			(Ticked)		
Archive and view past games					(Ticked)
A basic friend messaging system		(Ticked)			

*The interviewee asked for clarification on what an ‘Adaptive Engine’ was. I told him it was an engine which changed how it was playing within the same game.

Diagrams of existing systems

Left: Flow chart of the game-by-game analysis system on a Lichess/Chess.com-like website

Right: Data Flow Diagram of the analysis side of a Lichess/Chess.com-like website



User Requirements

Required Features

- Provide a platform that allows the user to play a game of chess against the computer.
- Develop an engine strong enough to beat existing engines of 1000 ELO at least 95% of the time.
- Allow the user to change the playing style of the engine, such that the change is noticeable to all users.
- Allow the user to decrease the strength of the engine so that the user can achieve a 50%-win rate against the engine.
- Store the past games of the user in a database on the server.
- Create an authentication system for logging in and signing up to the website.
- Create a platform suitable for a younger audience (simple interface and minimise complexity for the user)
- Add a campaign to guide the user through different playstyles.

Features that should be added

- Adjust the playing style of the engine so it plays human-like.
- Provide the user with an analysis of which playstyles they played best against in a shareable format.
- User customisation to the site and storage of preferences to appeal to younger people.

Desirable Features

- Add the ability to look back at past games and generate a shareable link to share to friends.
- Email verification for logging in.
- View games after they have been completed.

Features that will not be added

- Higher level search algorithms such as Monte Carlo tree searching.
- Any form of neural networks or other types of AI.
- Move-by-move analysis of the users' games.

Objectives

The timings for each group of objectives should be followed, but some slack is expected. Some objectives overlap, as they may be completed concurrently with other objectives.

1. Create a signup and login system. (After Mocks – End of School Year)
 - If the user isn't logged in, they will be redirected to the login page.
 - The user will have to input an email, a name and password.
 - The user's password must be complex (special character, uppercase, lowercase, at least 8 characters) for the form to be submitted.
 - The user will have to re-enter their password to confirm it has been inputted correctly.
 - The user's email must be unique.
 - The user's password must be stored using a suitable hashing and salting algorithm.
 - When the user logs in, the server will give the user a cookie which has been signed by the server using a suitable dual key algorithm.
 - Checks should be done on the user data client-side and server-side to prevent unnecessary requests and from the user from bypassing checks by sending their own requests to the server directly.

2. Create a customisable chess engine. (Start of Summer holidays – 10th September)
 - The chess engine must be played within the browser, locally.
 - The engine must use a low amount of memory and CPU time to prevent the site from freezing.
 - The engine must play to a standard to beat players up the level of myself (~1200 ELO).
 - The engine must be able to play down to users who have only recently started playing chess, and still give even games.
 - More customisation settings should be given, at least an aggression setting and a setting for how well the engine positions its pieces.
 - There should also be a GUI which provides the user with an interface to adjust the settings of the engine.
 - The engine should also have an adjustable ‘depth’ value.
3. Create an adventure mode using the engine customisations. (1st September – 1st October)
 - The user should be able to play an adventure mode with a short story interspersed with chess games which act as battles/fights/opponents.
 - The adventure mode should be easy for most players, but losses should be expected sometimes.
 - The adventure mode should contain the user’s display name in speech.
 - The user’s current level should be saved automatically when the browser is closed, or the user switches computers.
 - Once the user finishes the adventure, they should have access to a statistics sheet.
 - The sheet should be downloadable to the user for sharing.
 - All games played on the adventure or otherwise should be stored on the server and reviewed by the user at any time move-by-move.
 - The games should be shareable to non-logged in users, but only viewed if they have a shareable link created by the person who played the game, or they are the person who played the game.
4. Create a REST API between the database and the website. (15th September – 5th October)
 - The API should be authenticated and authorised for each resource which needs to be kept protected.
 - The API should allow the user to change their name or delete their account if requested.
 - The API should respond to invalid requests with the correct HTTP 4xx error in most cases and all common cases (it’s usually unreasonable to not run into any 5xx errors when dealing with many routes).
 - The API should allow an admin user to access all routes on the server and all data in the database (apart from password hashes and salts, these should be NEVER sent on external HTTP requests).
 - The API should make SQL requests to a database for the appropriate resources.

Technical Research

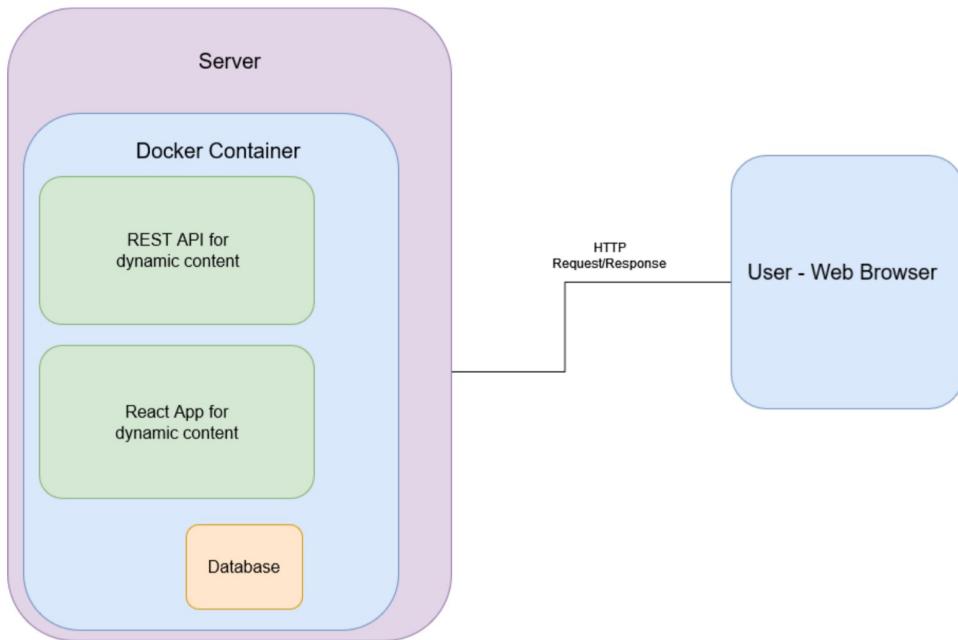
Almost all my research was conducted on the website <https://www.chessprogramming.org> which provides an extensive list of algorithms for creating a chess engine in many ways and contains the basic ideas for data structures which I based my TypeScript implementation on some of the ideas talked about on the wiki.

For setting up the Docker container, I used the documentation on the Docker website here <https://docs.docker.com/desktop/>.

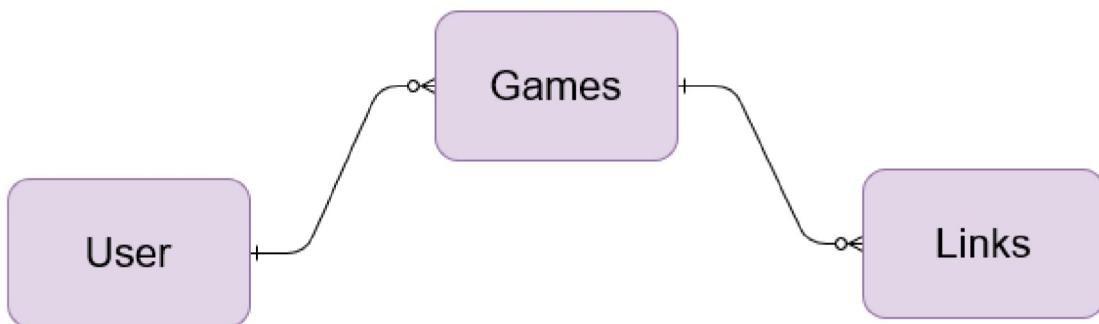
Similarly, I used the Flask documentation for the Flask app <https://flask.palletsprojects.com/en/3.0.x/>.

Problem Modelling

Diagram showing the general systems which will be included in the project.



Basic Entity Relationship Diagram for the database for the most important tables



Hardware and Software Requirements

I want to keep hardware and software requirements for the client and the end user as low as possible to ensure a range of users can use the project and a range of clients can host the project. The user must have an up-to-date browser to support some of the features that my project uses, but this should generally be met by almost everyone. As part of my objectives, I must not use anything which restricts the project to a specific desktop browser, or any feature which is only very partially supported. For the client, I would like for the project to be hosted in the cloud, preferably cheaply, below a few pounds a month running 24/7.

2 Design

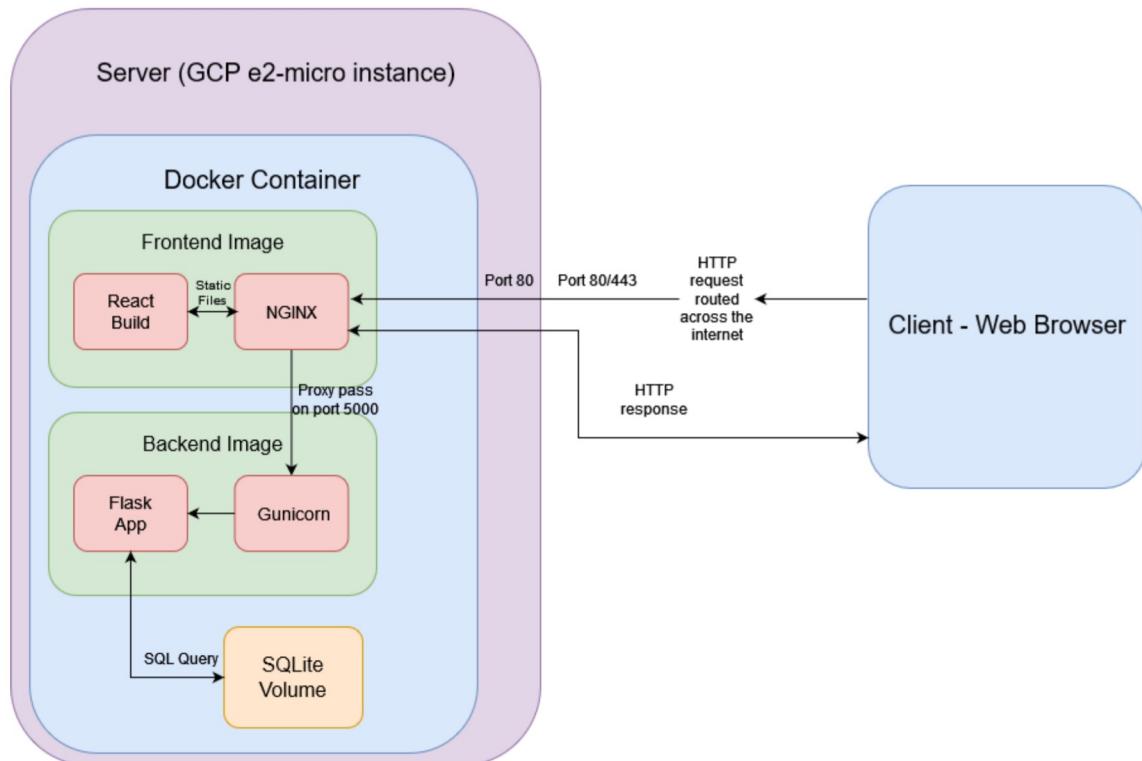
High Level Overview

The chess learning site is comprised of two main parts. The frontend and the backend. The backend will handle communication with the client (frontend) using a REST API, mapping the HTTP methods to the database CRUD operations. The backend will also handle authentication logic, including my own token signing algorithm for authentication and password salting and hashing. The backend will be written in Python, using the Flask framework, as I've used it in the past, with a serverless SQLite database as the webserver won't be receiving enough traffic to necessitate a dedicated database server. I will use gunicorn as the WSGI server for my Flask app which acts as a gateway for requests.

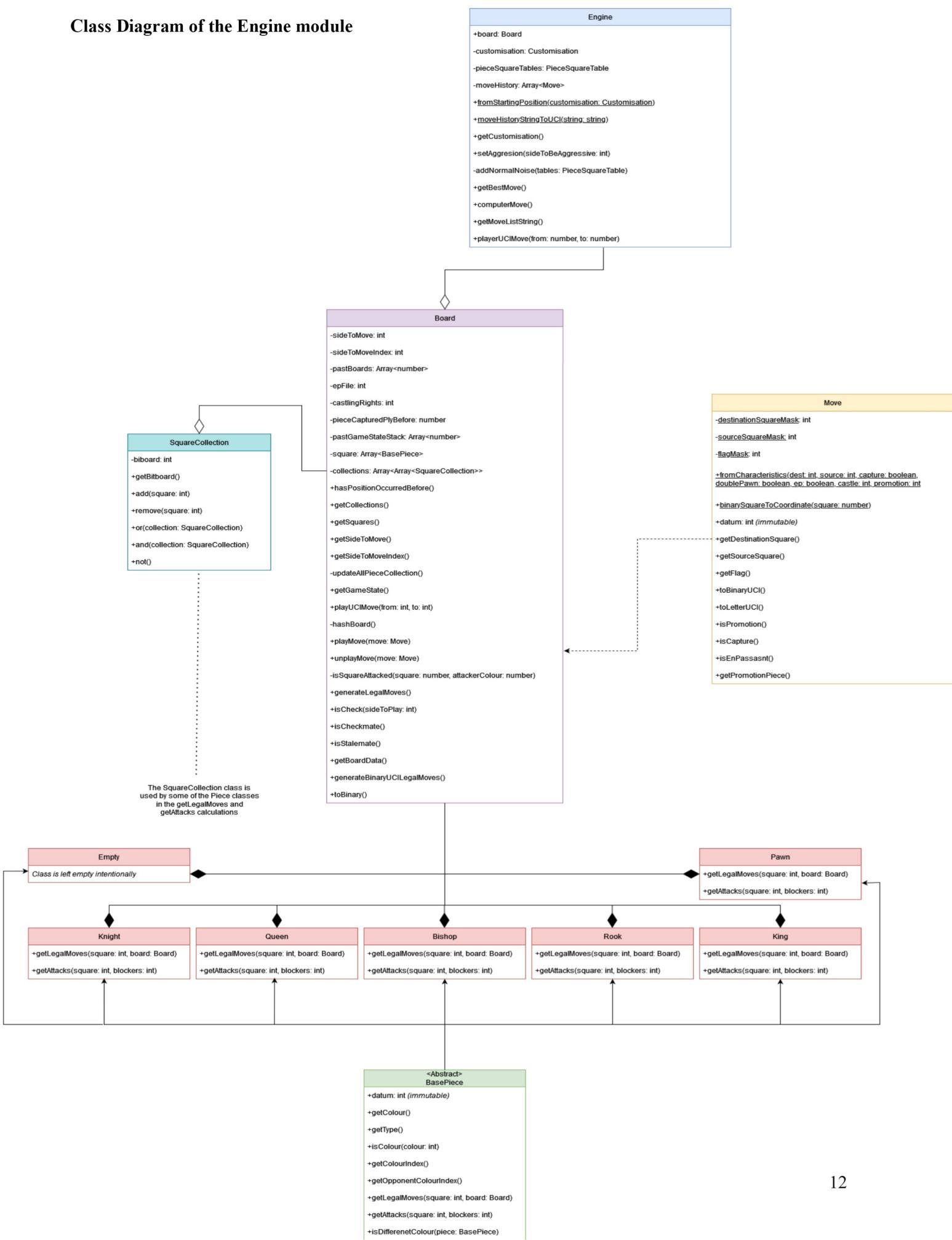
The bulk of my project sits in the frontend, which is a React App, written with TypeScript with class components. I opted to use TypeScript over JavaScript, as TypeScript is strongly typed, so I will run into less errors caused by mismatching types as my project grew larger, especially for the more delicate code in the chess engine. Routing for the React app will be done in browser for a smoother experience, as I'm not worried about the SEO falloff for using a single-paged app. The chess engine is fully separated from the React app, and the only communication between the engine and the React app is through the Engine class which acts as an interface between the two modules.

The frontend and backend are run as Docker images, with the backend container running the Flask app and the WSGI server. The backend container first builds the React app, and then starts NGINX which proxies API traffic to the backend container, and serves the static files of the React app. I use a persistent volume for the SQLite database and expose the NGINX server to the host machine's HTTP port (80), where the project can be accessed. I chose to break the project into Docker images so I wouldn't have any operating system differences, as I developed the project on my home Windows 11 computer, but when I needed to make the project accessible to anyone's computer, I hosted the project on a Google Cloud Compute Engine e2-micro VM, running on Debian. Using Docker also makes managing lots of services much easier, so I chose to use it for this project.

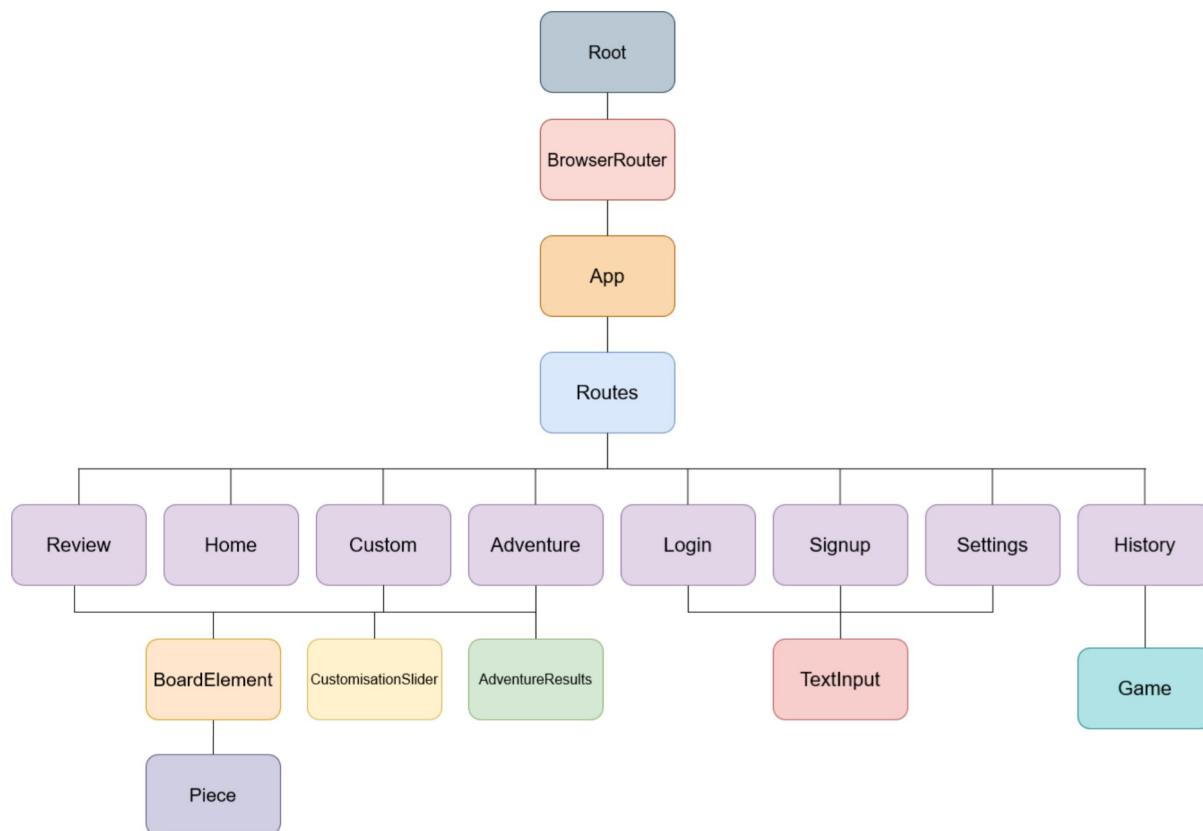
High level project architecture diagram – Excludes DNS routing and the Cloudflare proxy as they are irrelevant to the project itself.



Class Diagram of the Engine module



.Hierarchy Chart of React app components – (classes which inherit from `React.Component`)



The light purple components in the long line represent a route in the browser router. I will give a quick overview of what goes on in each route.

Review – Where the user can view a past game. The route takes the URL query parameters `userid` and `gameid`, which must be given to make the API request to the server of the game. We need the `BoardElement` class so we can look through the moves on the board.

Home – The user can select from the `Adventure`, `Custom`, `History`, and `Settings` routes.

Custom – The user can change the settings for the engine, and then play a game against the engine. We need the `BoardElement` and `CustomisationSlider` elements for this.

Adventure – This is the main feature for the project. The user plays through a short story intertwined with chess games against enemies. Once the user completes the campaign, they can see their statistics displayed in the `AdventureResults` component. We will also need the `BoardElement` component for playing the games against the enemies.

Login – A form to login. The user will be redirected to this route if they are not logged in. The `TextInput` component is needed for the form.

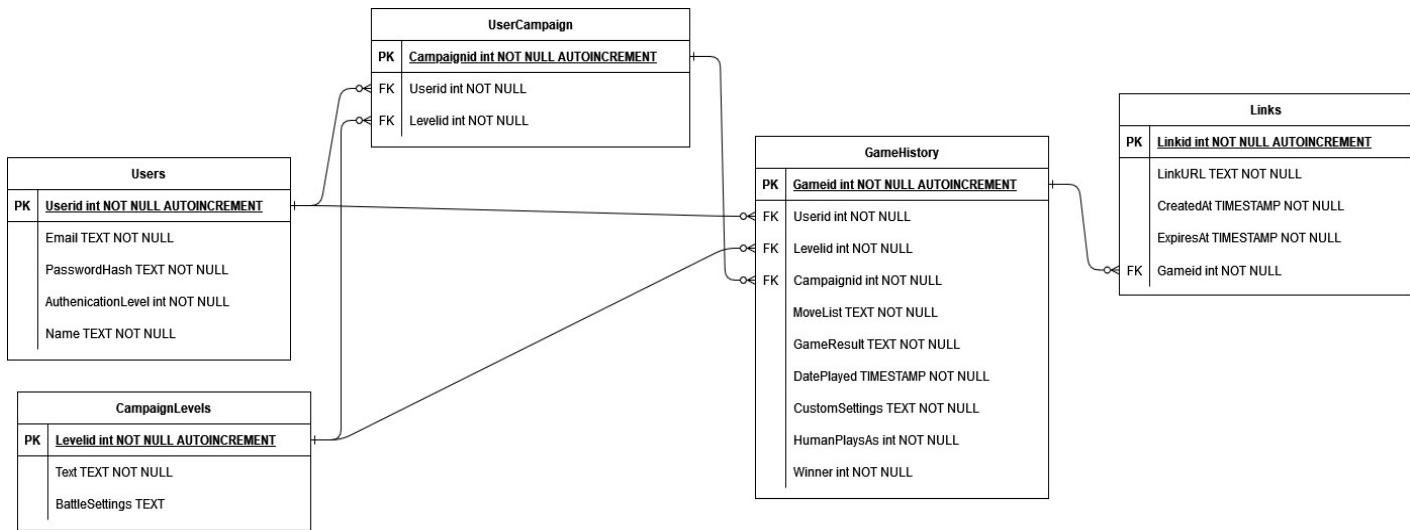
Signup – A form for signing up to the website. The `TextInput` components are also used in the form.

Settings – A route where the user can change their display name (which they will input in the `TextInput` component) or delete their account.

History – A route showing a list of all the games the user has played. Each element in the list is represented with a `Game` component. A link to the `Review` route of that game can be clicked for each game.

Database Design

Entity Relationship Diagram



Overview of SQL Queries Used

Question marks represent values which have been parameterised.

This query creates new links in the Link table. The LinkURL is a randomly generated Base64 string, which is inserted alongside timestamps which control when the Link expires. By storing the link in the database, the actual URL doesn't contain any information about the user or game, so when the link expires, it becomes useless. The SQL DATETIME function creates a UNIX timestamp that is one day ahead of the current time, and the CURRENT_TIMESTAMP variable inserts the current timestamp.

```
INSERT INTO Links (LinkURL, CreatedAt, ExpiresAt, Gameid)
Values(?, CURRENT_TIMESTAMP, DATETIME('now', '+1 day'), ?)
```

This is the query used to get information for a redirect when a user navigates to a shareable link (starting with /s/). The query uses an inner join between the Links table and the GameHistory table through the Game's ID, to get the user id of the person who shared the game for the redirect to /review?gameid={game id here}&userid={user id here}

```
SELECT Links.*, GameHistory.UserId
FROM Links
INNER JOIN GameHistory
ON Links.Gameid = GameHistory.Gameid
WHERE Links.LinkURL = ?
```

We select users' information from the user table in the login process to check the password hash, in the signup process to check for an existing account, and every single time the website is loaded. I also use an INNER JOIN to the UserCampaign table to get the current level id, so when the user plays the adventure mode, the API request will query the current level correctly. The user's id is also used for all API queries relating to them, so the client must have this information about the user when the website is first loaded.

```
SELECT Users.* , UserCampaign.Levelid
FROM Users
INNER JOIN UserCampaign
ON Users.UserId = UserCampaign.UserId
WHERE Users.Email = ?
```

This is a simple query just used to update the user's adventure level when they beat the previous level.

```
UPDATE UserCampaign SET Levelid = ? WHERE UserId = ?
```

This query selects all columns from the game history under one user's identifier and orders them from most recent to least. This is used in the API request for the History route.

```
SELECT * FROM GameHistory WHERE UserId = ? ORDER BY DatePlayed DESC
```

Normalisation

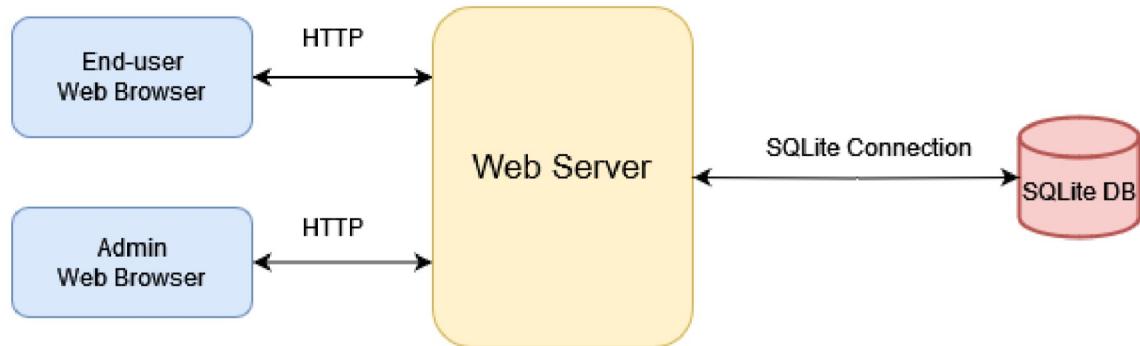
I've removed all partial and transitive dependencies in the database, and all non-key attributes depend on the primary key, the whole primary key and nothing but the primary key. It however can be argued that some of the data is not atomic. The GameHistory.CustomSettings and CampaignLevels.BattleSettings both contain stringified JSON strings, which obviously is not atomic. However, the data is not being queried within the JSON string, and I don't want to have to give the backend an understanding of the frontend, as that would remove a lot of the encapsulation between the frontend and backend modules, which I value as more important than the atomicity of the data, especially when the data does not have to be read or modified in the backend.

Cascade

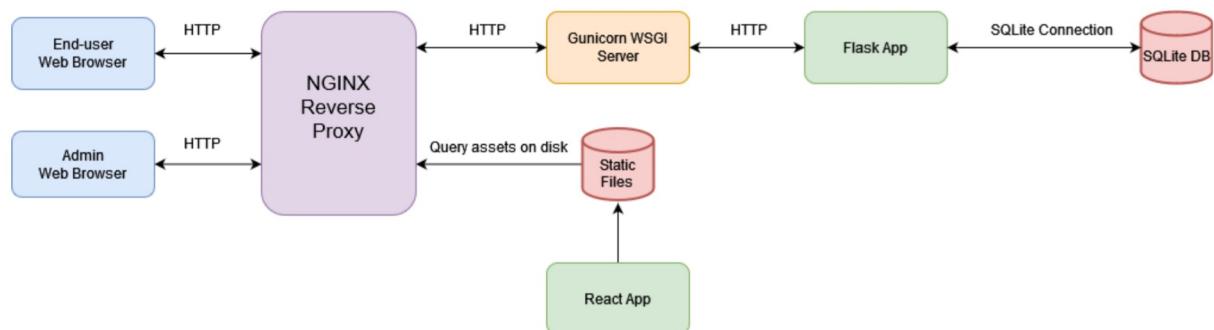
The only record which would be reasonably deleted with a DELETE query would be a user record. Cascade delete should be toggled for the UserCampaign table, as the campaign is meaningless without the userid being valid. The GameHistory records for the deleted records should also be deleted, which then would delete the links in the Links table which correspond to the deleted games.

Data Flow Diagrams

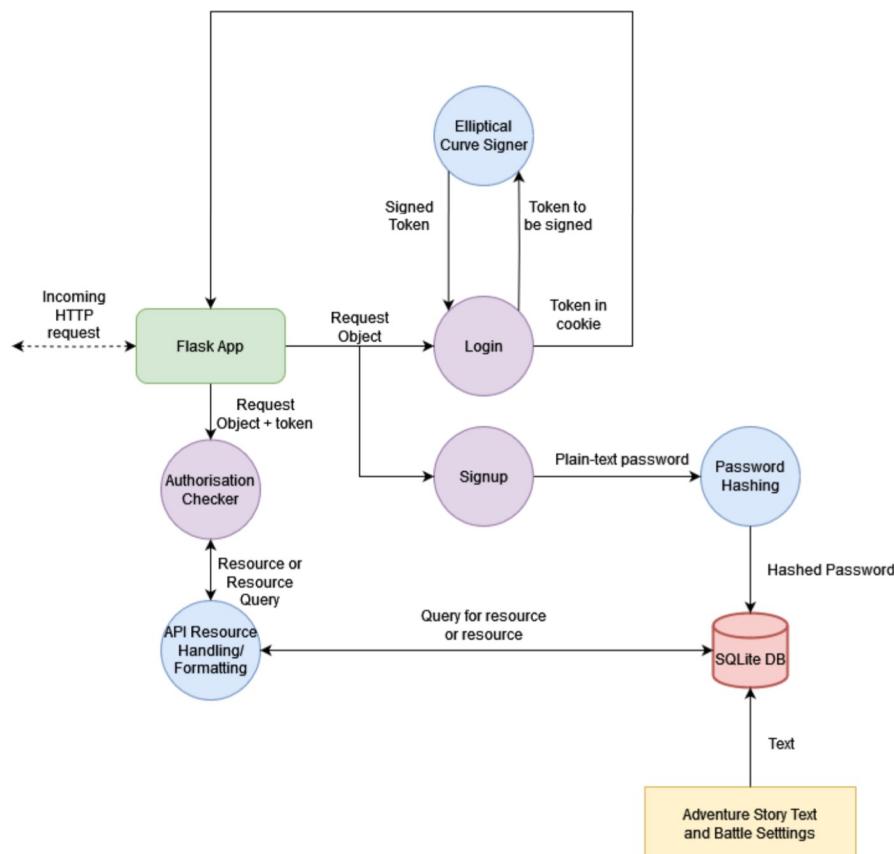
Level 0



Level 1



Level 2 (Focused on the Backend)



Data Structures

Most data in the engine is stored using integers, because we are restricted by the browser with how much memory we can use, and we get ability to use very fast operations such as bitwise AND, OR, XOR, and left and right shifts, which I use extensively in the engine, especially for functions which could be called tens of millions of times. Here is a list of some of the opaquer data structures used exclusively in the engine.

SquareCollection – Bitboard (*bitfield*)

This class is an interface in front of a 64-bit unsigned integer which represents some attribute for each of the square on a chess board. This is used in the `<Board>.collections` arrays, where each side of the board an array of SquareCollections for each piece representing if a specific piece is present on each square. This is used on top of the `<Board>.squares` array, as we can use bitwise operations for example to calculate captures, by ANDing the bitboard of attacks and the bitboard of enemy pieces, which is much faster than looping over the entire list of squares. Before when I just had the squares array, the function for calculating if the king was in check would be called millions of times in the PERF tests, and it would be incredibly slow, as the entire board would have to be iterated over each time. With the bitboard, pins can be calculated easier, so less psuedolegal moves must be verified by playing them on the board, and the bitboard can calculate if the king is in check by just ANDing the attacks of enemy pieces with the bitboard of the king.

Move – Bitfield

Moves are encoded into a 16-bit unsigned integer using the bitfield described in the Encoding Moves section of the chessprogrammng wiki. This reduces their memory consumption and allows for easy calculating by using binary masks for looking at a specific property of a move. The corresponding move for each flag can be found on the wiki, which I have used.

Destination Square						Source Square						Flags			
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Piece – Bitfield

Again, I'm using a bitfield for the piece, as it allows for easy comparison, and helps with the hashing of the board, as each square has a binary value for each piece. The 3 least significant bits are used to store the type of the piece (queen, king, bishop, etc.) and the 4th bit represents if the piece is black, and the 5th bit represents if the piece is white. This allows for easy checks if a piece is a specific colour, as we can use the mask `0b11000` which gives 8 if the piece is black and 16 if the piece is white and 0 if the square is empty.

Game State – Bitfield

When a move is played on the board, important attributes of the board which are about to change are stored into the game state, which is then stored in the game state stack. This allows the moves to be unplayed, and decreases the memory used by the board if we used a larger data structure like a dictionary. The attributes we store are the current side to move, the file which en passant can happen on, castling rights, and the piece captured the move before. With all this information we can unplay the entire history of moves played on the board.

Past Game States/Boards – Stacks

The past game states and past board hashes are stored in a stack. The same board is used while moves are played on the board while the engine is calculating, so the engine must be able to reverse all the moves played. The past game states stack allows this by storing the most recent game state on the top of the stack. If a new move is played, the current game state is pushed on the top, and if a move needs to be unplayed, then the top game state is popped and set as the current game state. The past boards stack also works in a similar fashion, but instead of holding the game state, it holds hashes of the past board positions, which are used to detect draws due to repetition. A stack is the best structure as searching for the most recent game state is always on the top of the stack, so it can be accessed in constant time.

Squares – Array

One way the board is represented is through a 1D array which represents each square on the board. Using a 1D array over a 2D array gives us more freedom to pre-program the ‘offsets’ for each piece’s movement. For example, to move a white pawn up a square, we add 8 to its index, and a capture represents the offsets of 7 and 9 for the left hand and right hand captures accordingly. We can also give repeated offsets of multiples of 8 and 1 for sliding pieces up and right accordingly, and 7 and 9 for the leading and adjacent diagonal. When pieces wrap around, we introduce checks to prevent this, for example a pawn on the 24th index cannot capture a piece on index 31. The array has a fixed length of 64, with each element being a 5-bit integer representing a piece using the bitfield discussed earlier.

56	57	58	59	60	61	62	63
48	49	50	51	52	53	54	55
40	41	42	43	44	45	46	47
32	33	34	35	36	37	38	39
24	25	26	27	28	29	30	31
16	17	18	19	20	21	22	23
8	9	10	11	12	13	14	15
0	1	2	3	4	5	6	7

Collections – Array

This is the second way pieces are stored on the board, which uses a 2D array and stores, not the pieces and square indexes, but a bitboard for each type of piece. Each 64-bit bitboard has a ‘1’ on the squares where the piece it’s representing exists, and a ‘0’ otherwise. We have a bitboard for the white king, black king, white pawn, black pawn, etc. for all the pieces. There is also an additional ‘white all’ bitboard and ‘black all’ bitboard which has a ‘1’ if any piece is on the square and ‘0’ otherwise. Using the methods on the Piece classes and SquareCollection class, we can write very clear code even though the collections have been abstracted so much. The ‘Pieces’ enum lets us index the Collections array with identifiers instead of just integers, making the code very readable.

Removing a piece from the collections square after a piece has moved.

```
Collections[piece.getType()][piece.getColourIndex()].remove(move.getSourceSquare())
```

Adding a Rook to square 3 after kingside castling has occurred (white).

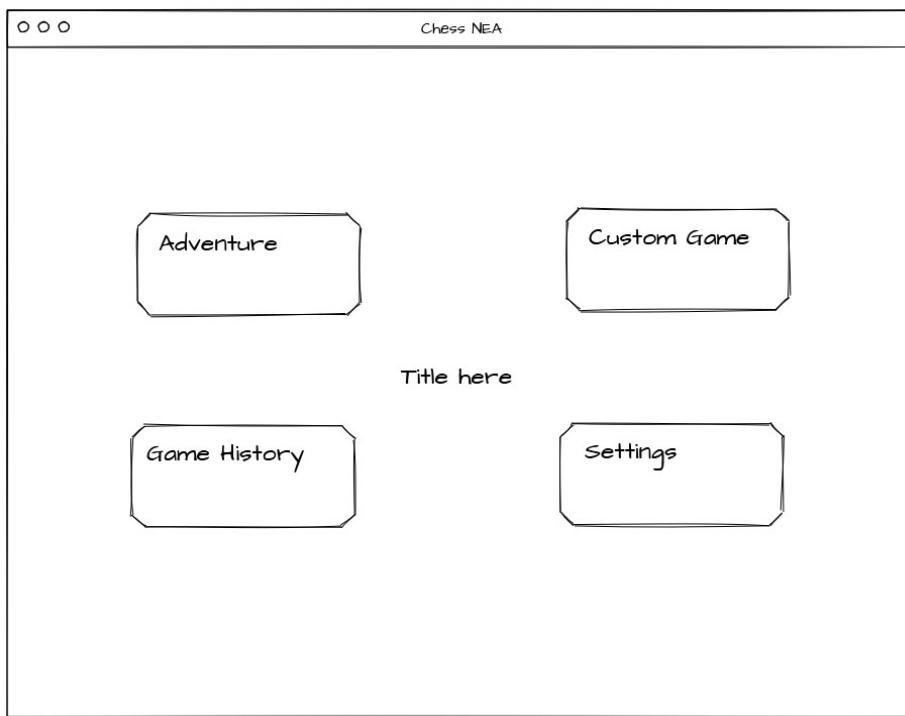
```
Collections[Pieces.Rook][Pieces.white].add(3)
```

Calculating captures for a non-blockable piece (king, knight, pawn) after attacks have been calculated, using the SquareCollection’s bitwise AND method with the opponent’s pieces bitboard.
`Captures ← Collections[Pieces.all][OpponentColourIndex].and(attacks)`

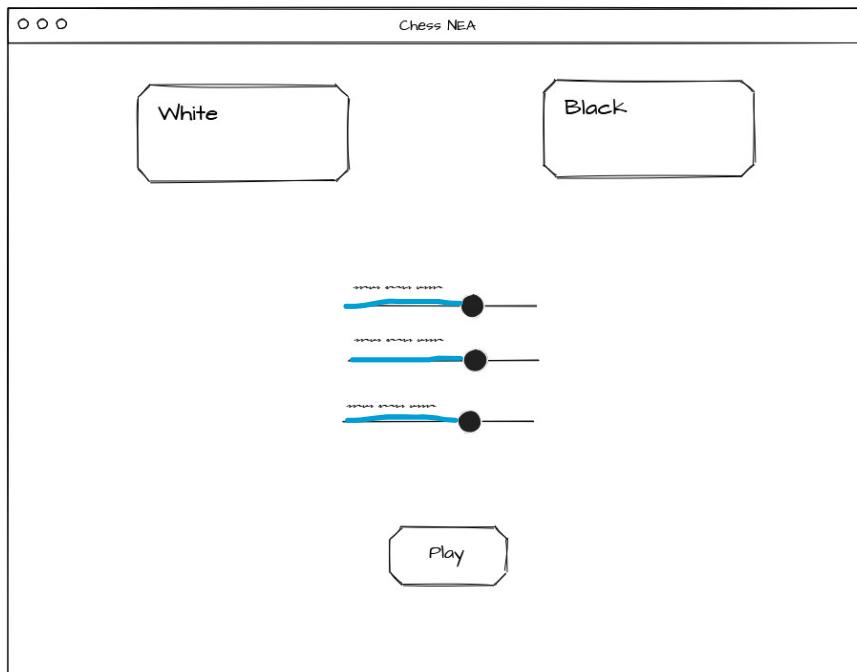
User Interface

Prototype Screen Designs

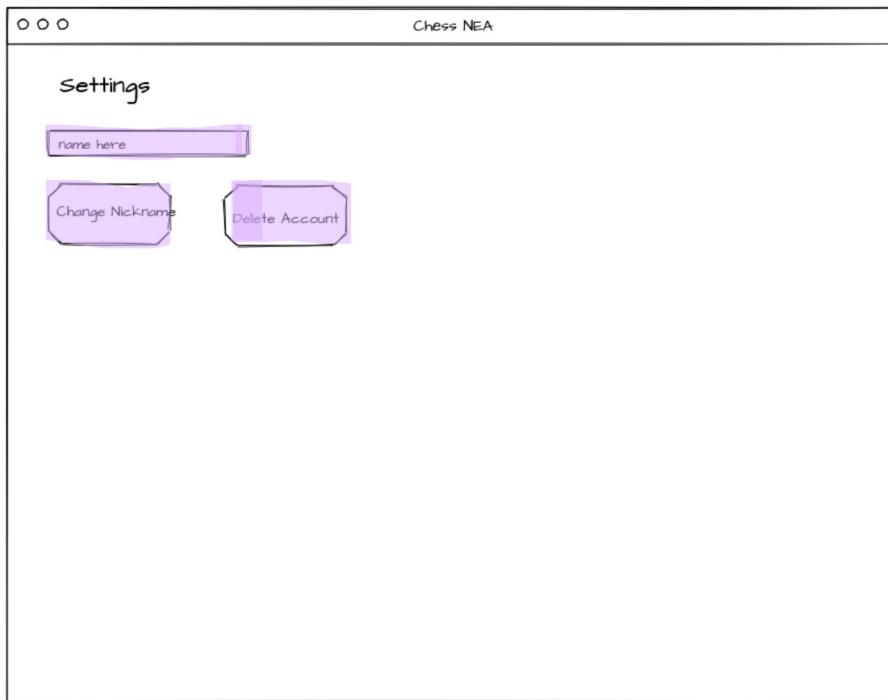
Home Route – This is the home screen where the user can choose from the given options on what to do. It's a simple layout, I'll make SVGs for each option, so they feel more interactive. I'm centring the options vertically so it's clear that this is a home page.



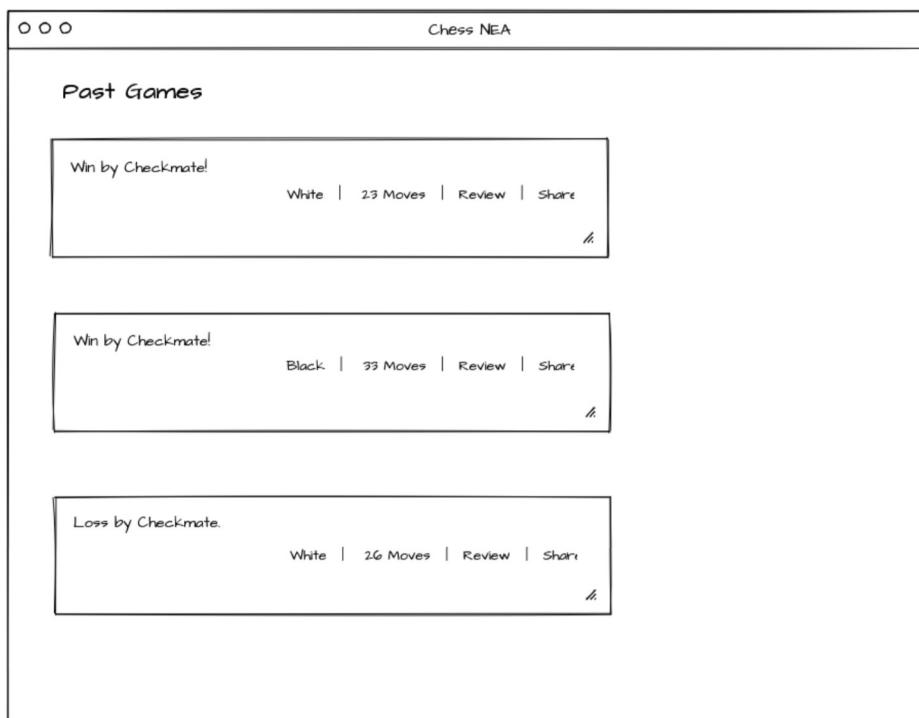
Custom Route – This is place where the user creates the custom game by moving the sliders. I don't know how many sliders I'll have, but it'll be about 3-5.



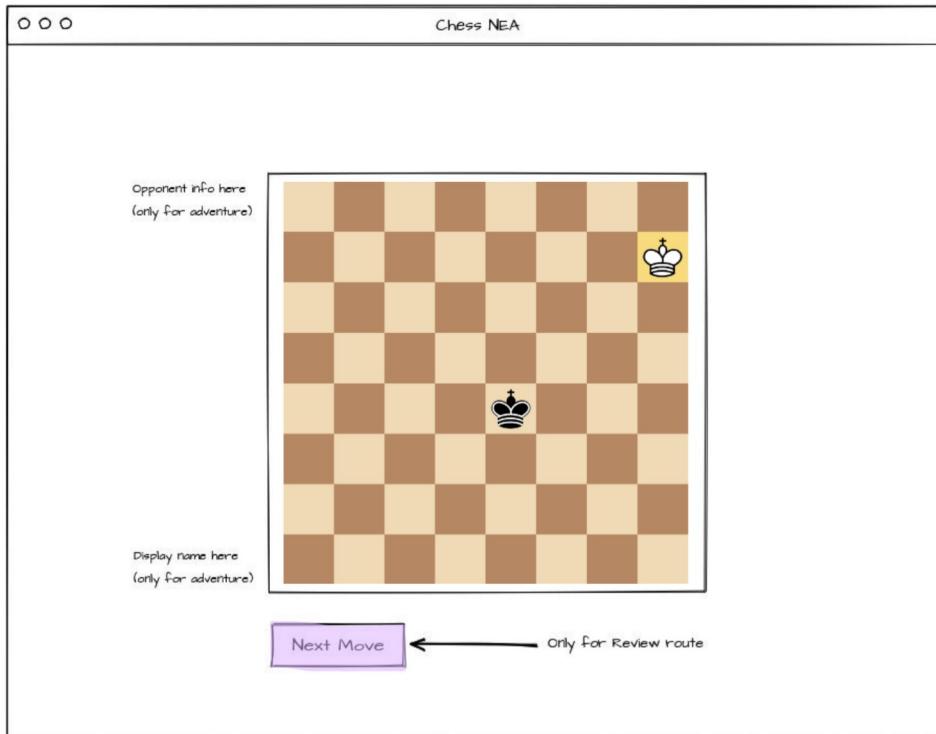
Settings – Settings for changing the user's name and deleting their account. I might have more options, but these are the main ones I want to include.



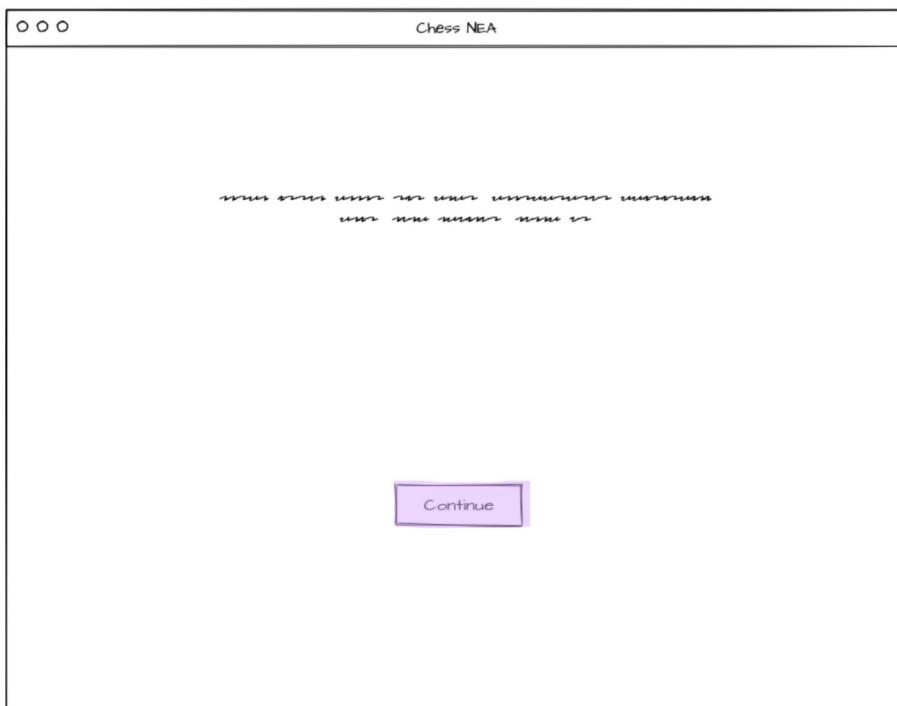
History – This is the route where the user can see a list of games played on their account. Clicking the review text will redirect them to the Review route. The top game should be the most recent.



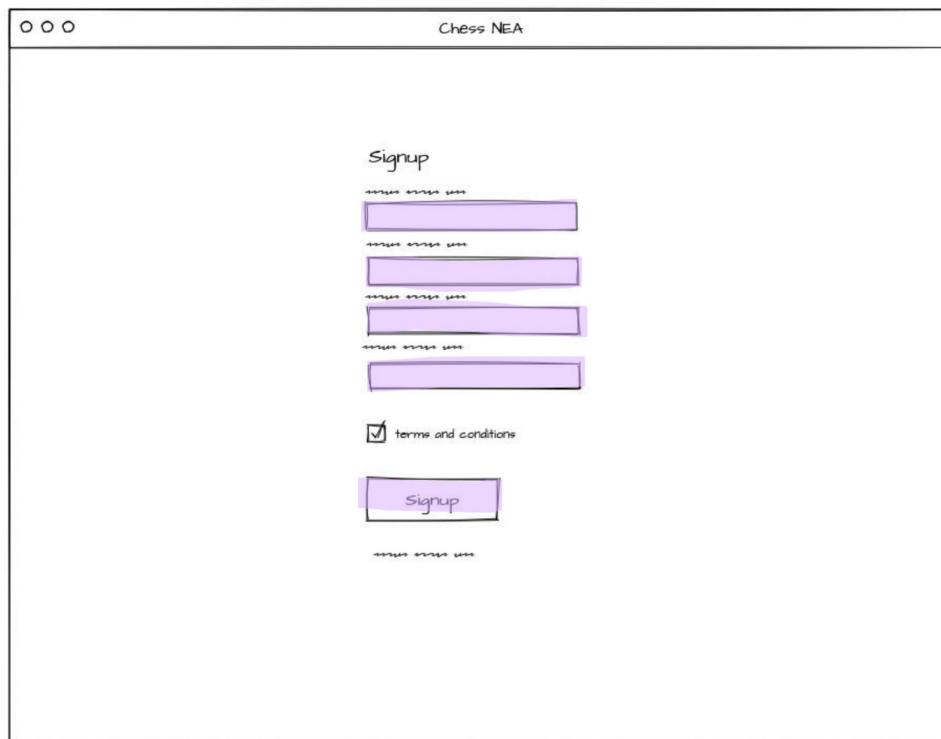
Review/Adventure/Custom playing game – These are the screens which contain a chess board. In the case of the review route, the board cannot be modified, only stepped through move-by-move using the Next Move button. With the adventure route, the text on the right will show the opponent and the user's name. In the custom mode none of this will be shown. An indicator of checkmate or stalemate should appear on the top left when the game ends.



Adventure going through the story – This screen will be visible when the user is playing the adventure mode and reading the story. The continue button will advance to the next piece of text, and eventually to a ‘fight’ (a chess game).

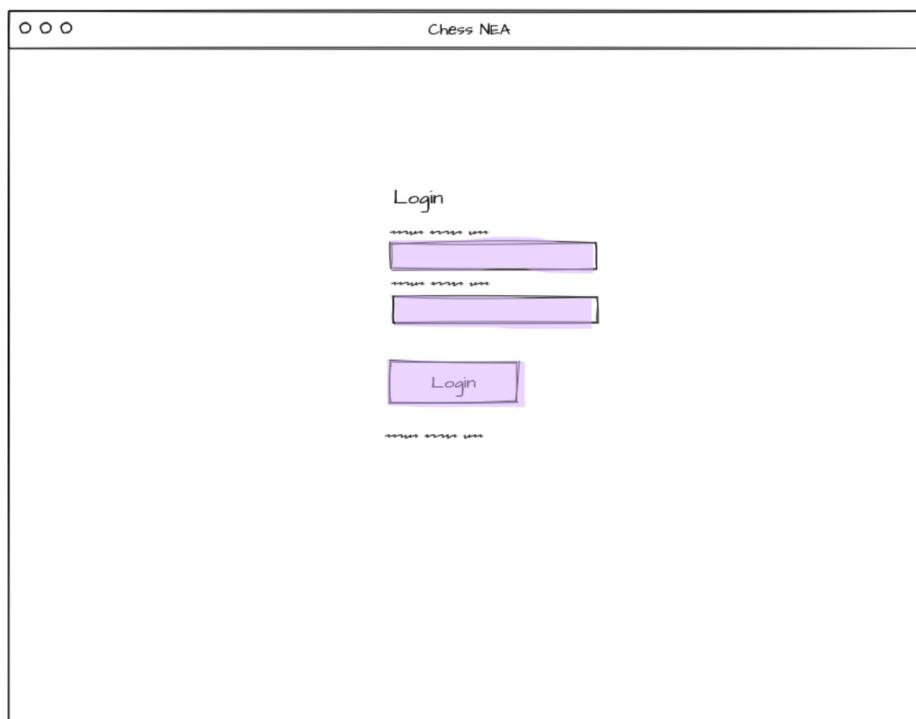


Signup – This screen is for the signup form. I've omitted the details to fill in, but they will be the email, name, password, and maybe one more. There might be a terms of conditions button for a terms and conditions and private policy to comply with data processing laws and similar legislation.



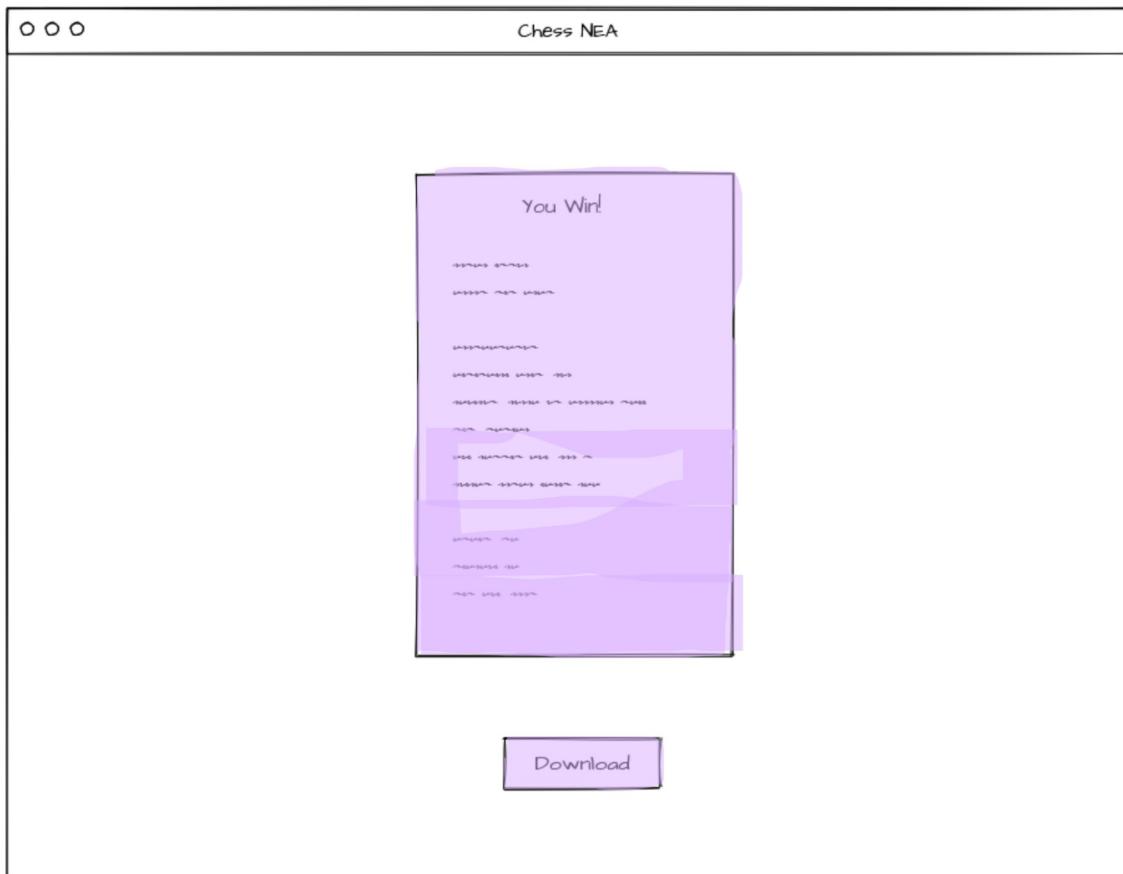
A wireframe of a mobile application screen titled "Chess NEA". The title bar includes three dots on the left and the text "Chess NEA" on the right. The main content area is labeled "Signup" at the top. It features four horizontal input fields, each with a placeholder consisting of five wavy lines. Below these fields is a checkbox labeled "terms and conditions" with a checked mark. At the bottom is a purple rectangular button labeled "Signup".

Login – Similar to above, but the two text boxes will be email and password. All these forms are centred both vertically and horizontally, as it makes the site seem more interactive, and suits better for a larger range of screen resolutions.



A wireframe of a mobile application screen titled "Chess NEA". The title bar includes three dots on the left and the text "Chess NEA" on the right. The main content area is labeled "Login" at the top. It features two horizontal input fields, each with a placeholder consisting of five wavy lines. Below these fields is a purple rectangular button labeled "Login".

Adventure Finished – The statistics sheet. I'm not sure what data I'm doing to collect (probably the same as the sliders on the Custom route). I also want to add a counter of some sort (games, moves, time), so the adventure becomes more like a competition, encouraging people to download the sheet and flaunt it to their friends.



General Design and Accessibility

As my project is aimed towards a younger audience, it is important the designs are modern, but also simple. I've chosen the Varela Round Google Font, as it is very readable, whilst still being distinctive. I'm also going to go on the larger side for fonts and buttons, to make the system easier to use, and provide responsive changes when the user presses a button by changing the button's colour, so it's clear that it has been pressed. I also will have a loading screen which is just the words "loading...", which will display if a route is taking a long time to lazy load, or if the client is waiting for an API request and it has no content to show.

I added the HTML attribute tabindex to the buttons on the Home page, the buttons on the Custom page, and the 'share' text on the History page. I had to do this as all these elements were generic elements which were clickable, so adding the tabindex property would allow users to navigate and use the website only using a keyboard.

More changes will be made to the interface in testing if any of my testers have problems navigating the system, although I will be more hesitant to make changes which mean rewriting a large part of the routing code. I have no accessibility plans for playing the moves on the board without a mouse, as click and drop behaviour alongside drag-and-drop behaviour is out of the scope of this project.

Algorithms

Elliptic Curve Signing Algorithm

An elliptic curve defined on the field F_p is represented by the equation $y^2 = x^3 + ax + b \pmod{p}$

The general steps for the signing algorithm are described in the US government document here <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-5.pdf>.

As I'm dealing with sensitive user information, it's important that I implemented the algorithm fully, as the document only outlines general guidance for the mathematical steps to ensure security, not the actual implementation which I created. The curve I used (secp256r1) was chosen from the document here <http://www.secg.org/sec2-v2.pdf>, but my implementation extends across all prime fields with $b \neq 0$ and order less than 2^{512} .

The point at infinity is the point at which the gradient of the curve at the point diverges to infinity. This will lead to divide-by-zero errors in our calculations, so instead we define the point at infinity to be $(0, 0)$, which does not lie on the elliptic curve given $b \neq 0$. We won't think about the curve, instead consider the algebraic group of points on the elliptic curve generated by applying a generator point to itself repeatedly using elliptic curve point addition. The generator point and the order of the group is given in the document, I only must worry about implementing the elliptic curve scalar multiplication.

To do scalar multiplication on a point, we just conduct point-to-point addition repeatedly.

$$sG = G + G + G + G + \dots + G \text{ (s times)}$$

This results in s total operations, giving the algorithm a time complexity of $O(s)$, which is too slow, as our scalar, s , ranges from 0 to 2^{256} . Instead, we can use the double-and-add algorithm, where we split the addition into powers of two. We can do this as associativity is guaranteed by the group axioms. Below is an example with $s = 59$.

$$59G = G + 2G + 8G + 16G + 32G$$

This means we are computing the same number of operations as the bit length of the scalar, s , giving a time complexity of $O(\log_2(s))$, which is much more suitable for larger numbers. We can implement this by creating a binary mask which represents a power of two and sliding the mask from the LSB of the scalar to the MSB, and doubling a point with each slide, adding this point onto the point total only if the mask matches at the specific bit of the scalar.

Pseudocode - & represents the bitwise AND operation, << represents the binary left shift operation and $+_{\text{E}}$ represents elliptic curve point addition, and \mathcal{O} represents the point at infinity

```

SUBROUTINE scalar_multiplication(Scalar, point)
    Mask ← 1
    Doubler ← point
    Total ←  $\mathcal{O}$  # Point at infinity as  $\forall p \in \langle g \rangle, p + \mathcal{O} = p$ 

    WHILE Scalar >= Mask
        IF Mask & Scalar THEN
            Total ← Total  $+_{\text{E}}$  Doubler
            Doubler ← Doubler  $+_{\text{E}}$  Doubler
            Mask ← Mask << 1
        ENDWHILE
        RETURN TOTAL
ENDSUBROUTINE

```

The point-to-point addition, given two points (A, B) and (C, D), can be calculated using these two equations well known elliptic equations.

$$X = \lambda^2 - A - C \quad \text{and} \quad Y = \lambda A - \lambda X - B$$

Where λ is the gradient of the line connecting the two points. These equations only work when the two points are not group inverses (so $A \neq -C$), otherwise the result is the point at infinity (the identity element). By the identity group axiom, if one of the points are the point at infinity, the result is the other point unchanged. To calculate the gradient, the simple rise-over-run equation words fine, but if the points are equal, then we will have to find the gradient using the derivative of the elliptic curve equation.

$$y^2 = x^3 + ax + b$$

$$\frac{dy}{dx} 2y = 3x^2 + a \quad (\text{using implicit differentiation})$$

$$\frac{dy}{dx} = \frac{3x^2 + a}{2y}$$

We then can use the three equations above.

Pseudocode – Let \mathcal{O} denote the point at infinity. Let $\%$ represent the modulo operator. Let $/_p$ represent the multiplicative inverse of two numbers modulo p , where p is the order of the field which the elliptic curve is defined in.

SUBROUTINE PointDouble(P)

```

IF P =  $\mathcal{O}$  THEN
    RETURN P
ENDIF
Gradient  $\leftarrow ((3 * P.X^2 + a) /_p (2 * P.Y)) \% p$ 
X  $\leftarrow (Gradient^2 - 2 * P.X) \% p$ 
Y  $\leftarrow (Gradient * (P.X - X) - P.Y) \% p$ 
RETURN (X, Y)
ENDSUBROUTINE
```

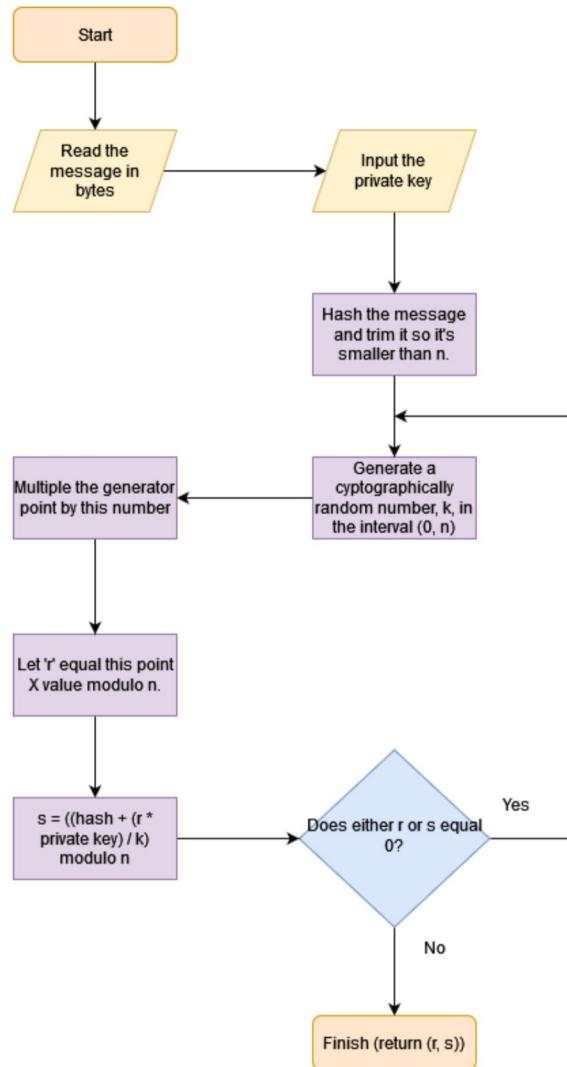
SUBROUTINE PointAddition(P, Q)

```

IF P =  $\mathcal{O}$  THEN
    RETURN Q
ENDIF
IF Q =  $\mathcal{O}$  THEN
    RETURN P
ENDIF
IF P.X = Q.X THEN
    IF P.Y = Q.Y THEN
        RETURN PointDouble(P) # Points are the same so just double P
    ELSE
        RETURN  $\mathcal{O}$  # Points are inverses, so return the point at infinity
    ENDIF
ENDIF
Gradient  $\leftarrow (P.Y - Q.Y) /_p (P.X - Q.X)$ 
X  $\leftarrow (Gradient^2 - P.X - Q.X) \% p$ 
Y  $\leftarrow (Gradient * (P.X - X) - P.Y) \% p$ 
RETURN (X, Y)
ENDSUBROUTINE
```

To do the actual signing algorithm, I implemented the steps on the US government document, using the functions I have created. Below is the flowchart of the algorithm. The value of n refers to the order of the group generated by the generator point of the elliptic curve. A similar method is used in the verify signature algorithm, using the public key instead of the private key, but more checks must be done, as we are dealing with user input instead. When multiplying a point by an integer, this refers to elliptic curve scalar multiplication.

Flowchart – Elliptic curve signing algorithm.



To generate a key pair, we just choose a random point in the group generated by G , by choosing a random value in the interval $(0, n)$ (this is the private key) and the point calculated is the public key. This operation is very hard to reverse, as it reduces to the discrete logarithm problem (as the group generated by G is cyclic so it must be abelian), just like other encryption algorithms such as RSA.

Pseudocode – Key pair generator. Let \times represent elliptic curve scalar multiplication, G represents the generator element.

```

PrivateKey ← Random.Range(1, n)
PublicKey ← PrivateKey × G
  
```

```

OUTPUT PrivateKey
OUTPUT PublicKey
  
```

For checking that a signed message has indeed been signed by the private key given, we can use the public key to check that the signature is valid for the message given. This subroutine is in the EllipticCurve class, which has read-only attributes defining, a, b, p, n, and such.

Pseudocode – Checks that the binary message matches the signature, given the public key. Let \times represent elliptic curve scalar multiplication. Let $/_n$ represent the multiplicative inverse of two numbers modulo n. Let $+_E$ represent elliptic curve point to point addition.

```
SUBROUTINE VerifySignature(Binary, Signature, PublicKeyInt)
    # The Public Key is actually a point, with the first nBitLength bits representing the x
    # coordinate and the last nBitsLength bits representing the y coordinate
    PK ← (PublicKeyInt >> nBitLength, PublicKeyInt & ((2^nBitLength -1)))

    IF PK = O THEN
        RETURN FALSE
    ENDIF

    # Check that the public key is on the elliptic curve in the field modulo p
    IF (PK.Y ** 2) % p ≠ (PK.X ** 3 + a * PK.X + b) % p THEN
        RETURN FALSE
    ENDIF

    # Check that the the order of the Public Key divides n over  $F_p$ 
    IF n × PK ≠ PK THEN
        RETURN FALSE
    ENDIF

    # Retrieve the signature point (r,s) from the signature
    R ← Signature & (2^nBitLength - 1)
    S ← Signature >> nBitLength

    # Check r, s ∈ (0, n)
    IF R < 1 OR R >= n OR S < 1 OR S >= n THEN
        RETURN FALSE
    ENDIF

    # Calculate and trim binary message hash into an integer
    Hash ← INT(SHA512(Binary)) >> (HashLength - (nBitLength + 1))

    # Now all parameters are safe, we can reverse the signing algorithm
    S⁻¹ ← 1 /n S
    U ← (Hash * S⁻¹) % n
    V ← (R * S⁻¹) % n
    RTest ← (U × G) +E (V × PK)

    IF R = (RTest.X % n) THEN
        RETURN TRUE
    ELSE
        RETURN FALSE
    ENDIF
ENDSUBROUTINE
```

Alpha-Beta Pruning

To find the best move in a position, we get all the moves in the position, and then play each move on the board. This is repeated recursively up to some depth value, where the board is then evaluated. To decide which branch of the move tree to follow we use the minimax algorithm. As chess is a zero-sum game (everything good for us is bad for our opponent and visa-versa), we are trying to maximise our own evaluation, while minimising our opponent's evaluation. Doing this by checking every node, we will have to check, m branches per each new branch d times, where m is the number of moves in a position, and d is the depth. This gives a time complexity of $O(m^d)$. We can optimise this by using alpha-beta pruning, where we 'prune' (stop searching) in branches which have a guaranteed worse evaluation than the current best move. The alpha value represents the best evaluation for the maximising player, and the beta value is the best evaluation for the minimising player. In the worst case, we never prune any branches (this would happen if we started searching for moves from worse to best). In the average case, half of the branches will be cut, giving a time complexity of $O(m^{d/2})$, but we can order the moves by looking at captures of high value pieces first, to increase alpha and decrease beta quicker, giving a better-than-average time complexity almost every time. To sort the moves, we use insertion sort, as it has the one the best performances for arrays with about 15 elements, and also sorts the moves in place, giving a space complexity of $O(1)$.

Pseudocode – Slightly different than the code in Search.ts. Doesn't include the customisation or the simplifyPosition function, only the alpha-beta pruning code. 'Infinity' represents the number which will return 'true' in the inequality, $\text{Infinity} > x$, for all x .

```

bestMove ← Move(0) # Empty dummy move
CONSTANT CHECKMATE_EVAL ← -999999999999
maxDepth ← 4

SUBROUTINE SearchDepth(board, depth, α, β)
    IF depth = 0 THEN
        RETURN EvaluateFunction(board)
    ENDIF
    Moves ← board.generateLegalMoves()
    IF LEN(Moves) = 0 THEN
        IF board.isCheck() THEN
            # Adding the depth means the computer evaluates quicker checkmates
            # worse than slow checkmates
            RETURN CHECKMATE_EVAL + depth
        ENDIF
        RETURN 0
    ENDIF
    EstimatedMoveOrder ← sortMoves(board, moves)
    FOR i ← 0 TO LEN(EstimatedMoveOrder) - 1
        Move ← Moves[EstimatedMoveOrder[i]]
        board.playMove(Move)
        EVALUATION ← -1 * SearchDepth(board, depth - 1, -β, -α)
        board.unplayMove(Move)

        IF EVALUATION ≥ β THEN
            IF depth = maxDepth THEN
                bestMove ← move
            ENDIF
        ENDIF
    ENDFOR
ENDSUBROUTINE

```

```

        ENDIF
        RETURN β
    ENDIF

    IF EVALUATION >= α THEN
        IF depth = maxDepth THEN
            bestMove ← move
        ENDIF
        α ← EVALUATION
    ENDIF
ENDFOR

RETURN α
ENDSUBROUTINE

```

searchDepth(board, maxDepth, -Infinity, Infnty)

OUTPUT bestMove

Board Hashing

A simple hashing algorithm to detect draws in the Past Game stack is employed. The past board stack shouldn't be long, only very rarely over 100 elements long. The limit for bitwise operations in JavaScript is 32 bits (without using the much slower Bigint type), which gives us 2^{32} possible hashes, making collisions almost impossible. However, we want to make sure that the hashing algorithm is chaotic, so slightly changing the initial conditions greatly changes the output, while still being deterministic. This is because the boards that we are hashing are very similar as only one move separates a pair of boards. We'll employ an XOR hashing algorithm, shifting each piece value by the index of the square it's currently on. We will have to binary shift the index to the left by one, as the maximum value of the index is 63, which will make the hash too big at 2^{63} . Now shifted, the maximum value is 31, which makes the hash have a size of 2^{31} , within the limit. We set the hash initially to the board's game state, so that contributes to the hash also.

Pseudocode – Actual code uses an array reduce function but results in the same output. Let ‘ \wedge ’ represent the XOR operation, ‘ $<<$ ’ and ‘ $>>$ ’ represents left and right shifts accordingly.

```

Hash ← BoardGameState
Index ← 0
FOR Piece IN BoardSquares
    NextHashPiece ← Piece << (Index >> 1)
    Hash ← NextHashPiece  $\wedge$  Hash
    Index ← Index + 1
ENDFOR

OUTPUT Hash

```

Adding Positional Weaknesses

When we want to make the engine play worse, one of the techniques that I will use is adding Gaussian noise to the piece square tables. These tables, which I found on the chessprogramming wiki, show the engine which squares are the best for each piece, encouraging the engine to make progress, castle, defend the king and move towards the opponent. By adding noise to these boards, the engine will play worse, which we want as part of the customisation.

To add Gaussian noise, we will sample from the normal distribution for each square, with a different variance, depending on how badly we want the engine to play, represented by the positionPlay customisation setting from 0 to 100. With 100 we want no variation, and with 0 we want maximum noise. Any variation more than 50 points will cause the engine to start sacrificing pieces to get onto certain squares, which we don't want, so we will say that a noise of +50 points should only happen in 1% of cases with maximum noise.

$$X \sim N(0, \sigma^2) \text{ with } P(X > 50) = 0.01$$

$$\frac{X - \mu}{\sigma} = z = 2.3263 \text{ (from an area right z score table at 0.01)}$$

$$\sigma = \frac{50}{2.3263} \approx 21.49 \approx 20 \text{ (as the mean is zero and } X \text{ is 50)}$$

To map our customisation setting to the standard deviation, we can use an inverse linear relationship.

$$f(x) = 20 - \frac{x}{5}$$

In the case where the positionPlay is 100, we will just return the piece square tables without noise.

To sample from X, we must use some transformation that takes a uniform random number in the interval (0, 1), and outputs a normally distributed number, as in JavaScript we can only generate uniform random numbers in the interval (0, 1). We can use the Box-Muller transform to create standard normal random numbers and multiply by the standard deviation to get our noise for each cell.

Pseudocode – Adding Gaussian noise to the piece square tables, using the Box-Muller transformation. Table cells are modified in place which I've shown.

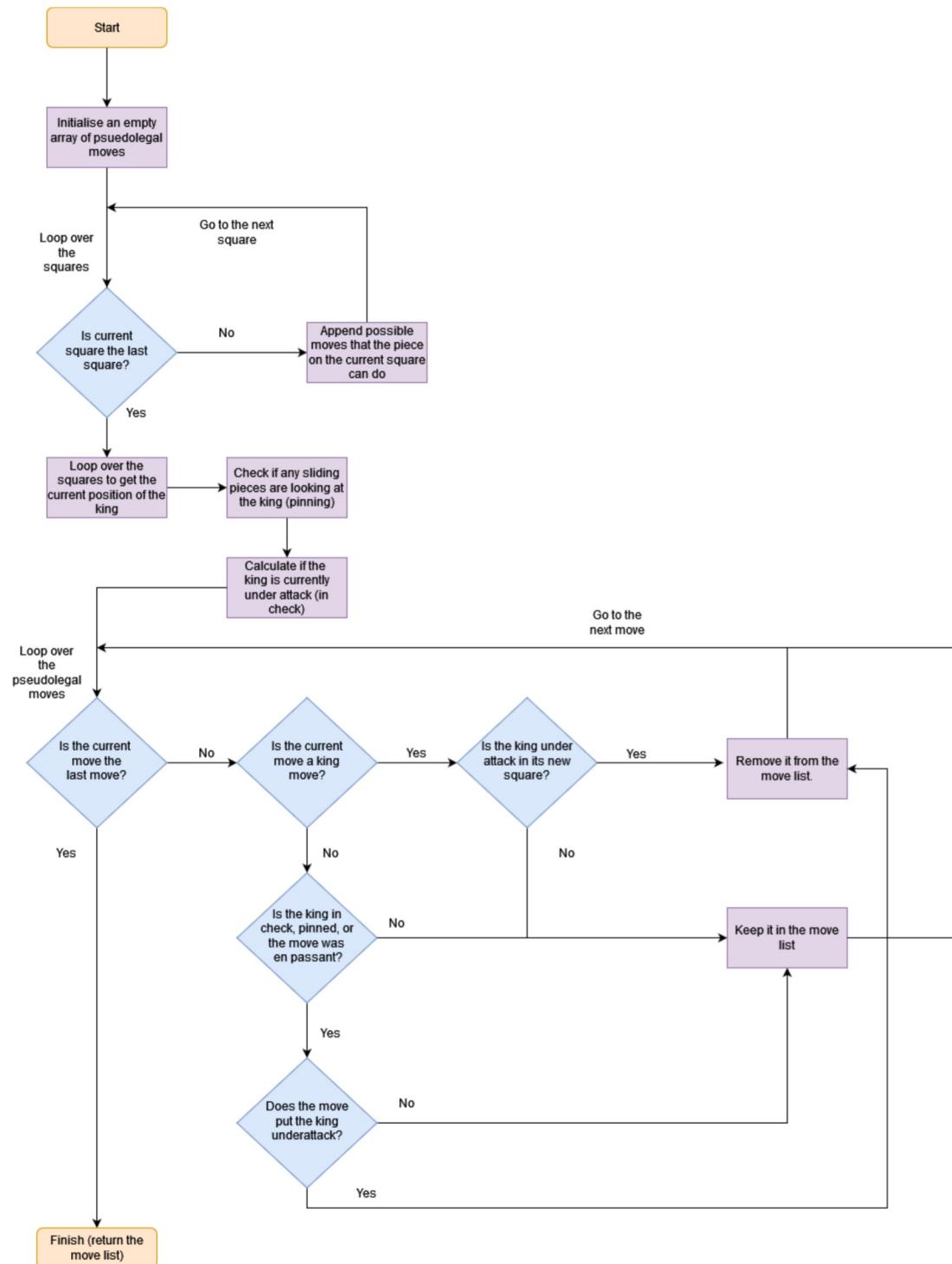
```

SUBROUTINE AddNormalNoise(Tables)
    IF positionalPlay = 100 THEN
        RETURN Tables
    ENDIF
    StandardDeviation ← 20 - (positionalPlay / 5)
    FOR TableGroup IN Tables
        FOR Table IN TableGroup
            FOR Cell IN Table
                U1 ← RANDOM() # Random number in the interval (0, 1)
                U2 ← RANDOM()
                X ← SQRT(-2 * NaturalLog(U1)) * COSINE(2π * U2)
                Table[Cell] ← Table[Cell] + (X * StandardDeviation)
            ENDFOR
        ENDFOR
    ENDSUBROUTINE

```

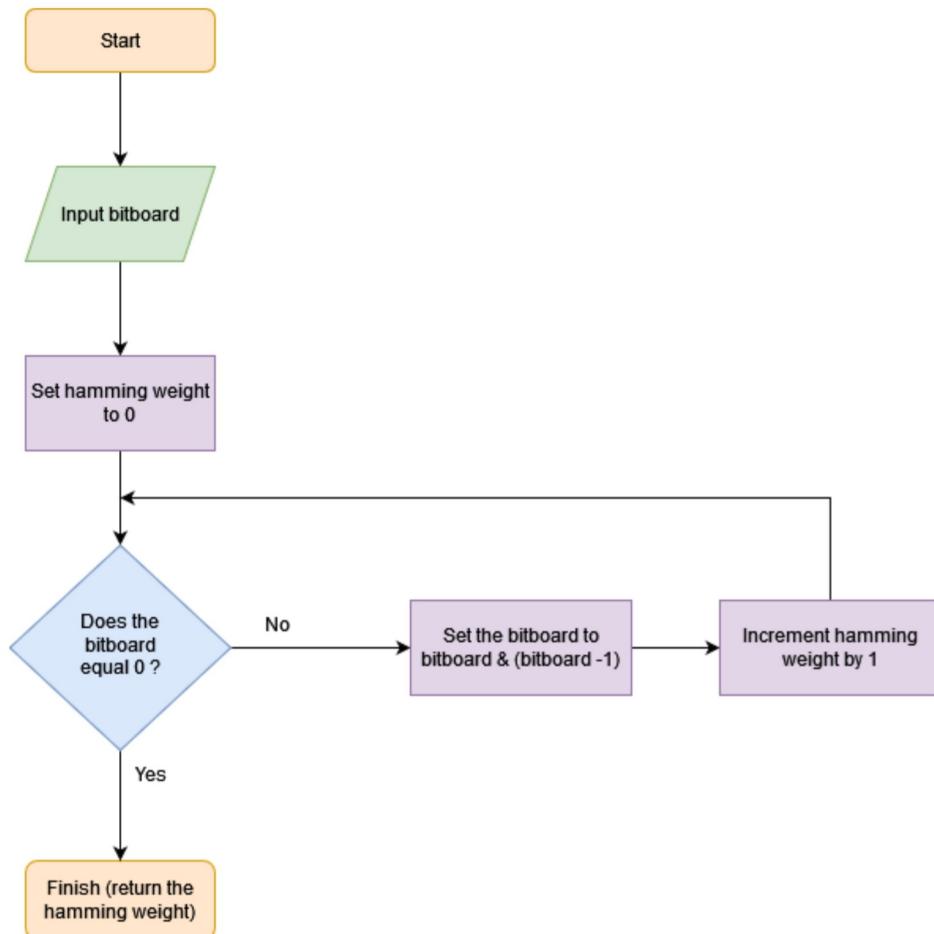
Move Generation Algorithm

This is the central algorithm for the engine, and it must be fast since it's called thousands and thousands of times. The algorithm looks for all 'pseudolegal moves' which are moves that could be played if we didn't care about the king being captured. This array is then filtered to remove any moves which put the king under attack either directly from the king moving, or indirectly through revealing an attack on the king. I calculate if a complicated move was legal or not by playing the move on the board, and seeing if the king is attacked in this future board. This function is computationally expensive, so the checks are done first to lower the number of calls to this function.



Piece Counting Algorithm

One of the major benefits of storing the pieces' locations in a bitboard is that we can easily calculate the number of pieces on a board for a specific piece or for all pieces. The way we can do this is calculate the Hamming weight of the bitboard, base 2. We can do this by creating a loop to continuously remove the lowest significant bit and increase the hamming weight counter by 1 for each loop.



Hardware and Security

As my project is built using Docker, it can run on any platform which supports Docker (Windows, Linux, MacOS, ChromeOS). As my project is a full stack web application, some expertise would be required to run the program from its source code, set up port forwarding, and domain setup. Using Docker eliminates most of the complications, and the primary target hardware will be the Google Cloud Platform which supports Docker. More specifically, the target product is the Compute Engine, with an e2-micro instance (10GB disk, 1GB RAM, 1 CPU), which can run the project smoothly. A main reason for choosing the e2-micro instance is that Google (as of October 2023) allows free access to one month of hours of an e2-micro instance per month (with some restrictions), so the client wouldn't have to pay any running costs or run their own dedicated server.

I've tested the project on Windows 11, 10, Debian Buster, Bullseye, and Bookworm with no problems. For the operating system within the Docker images, I chose Linux Alpine for the frontend container, due to the lower RAM usage and security benefits of a lightweight Linux distribution like Alpine, as I don't want to use go over the 1GB RAM limit of the instance, otherwise the container would use virtual memory, which would make it much slower. For the backend container, I chose Python's slim-bullseye Linux image, because I had some problems installing Gunicorn and Flask within the container using Alpine Linux. I only chose Bullseye Debian because it was the latest stable version when I started development (before June 2023).

My users will access the system using any up-to-date web browser. I won't be supporting mobile and touchscreen users, as a full responsive design is way too far outside of the scope for this project. The project will be built to the standards of the current ECMAScript standard, as many of the features such as JavaScript bigint and HTML5 canvas depend on the newest browser versions. I've tested the project on Gecko browsers (Firefox) and Chromium browsers (Google Chrome, Microsoft Edge) and I've only noticed slight disparities, most notably in the rendering of the SVGs on the Home route.

For security, as I'm dealing with user passwords, I need to protect the transmission and the storage of the passwords. In transmission, I'm using Cloudflare's DNS system to provide HTTPS encryption on the request body from the user to the Cloudflare server. For storage, I do not store the passwords in plaintext, instead I store the password hashes using the secure SHA-256 hashing algorithm. I also add a 'salt' to the passwords which prevents rainbow table attacks from precomputed hash tables, as each password gets added some random bits to the start of the password, adding uniqueness between precomputed hash values.

For authentication, I'm using a token-based system, where the server signs a proof of the client's id and authorisation level, which is then verified in each API request for user data. The tokens are signed using the elliptic curve secp256r1 which has been generally agreed on to be secure. Authorisation is managed by 'scoping' the tokens. The tokens are given an authorisation level, and possibly a user id or a game id to have access to. A default authorisation level 1 only gives access in the user scope in the API, and a level 0 only gives access to resources which match both the game id and user id scope. The admin authorisation level is 5 and can access all resources. The scopes and authorisation level cannot be changed by the client as this would change the hash of the message, invalidating the signature, which would result in the request being denied. A different public-key private-key pair can be created at any time, which invalidates all previous keys given out which could be done in the case of a leak of the private key. The k-value calculated when creating signatures is a cryptographically secure random number, preventing the extraction of the private key using mathematical analysis of the signatures generated.

The database is normalised to keep data integrity between tables and cascade delete is enabled for the relevant tables, so if a user is deleted, other records containing the user id are also deleted.

3 Testing

Test Plan

Before I start testing, I first want to break the testing up into different sections which reflect how the different modules of the code is programmed. This should give me less errors at once and let me be assured that important areas of the code are perfect.

Test 1 – Firstly, I must make a chessboard where the user can move pieces using the drag and drop action, where the state of the chessboard is stored programmatically so the engine can play against the user. In this test the board will be created using a CSS grid, to place the squares, and the pieces will be transformed to their position on the board as SVG elements. When dragging and dropping, the piece should follow the mouse and snap to the closest square. Evidence of this test will be done through screenshots and console outputs, and the dragging and drop behaviour can be later verified in other videoed tests.

Test 2 – For each piece on the board, the piece attacks a certain pattern of squares. Here bitboards are used to represent the set of squares which are attacked. This should provide foundation for the move generation algorithm. For each piece, the attacks should be calculated and output in a bitboard. The squares are stored in a 1D array, so different offsets are required to translate the piece depending on if it's close to the edge. For the knight and sliding pieces, these can be precomputed and hardcoded into the program, and read from at runtime. This is tested using a function to output the binary representation of the bitboard for each attack in each piece. A range of normal tests (centre of the board) and boundary tests (near the edge of the board or near the corners) should be completed.

Test 3 – The final move generation functions are the most complicated, but the most important part of this NEA. The tested functions are the piece's move generation methods, along with the Board's playMove and unplayMove methods. To provide the highest level of rigours testing, I will have an implementation of the PERFT test. Using test positions created and shared in chess programming forums, I can compare the number of leaf nodes in my move generation tree against the community agreed values. This should be done in the millions of nodes to prevent any edge cases from slipping through, as only one error could lead to errors in the evaluation of moves. Video evidence of the PERFT testing should be provided.

Test 4 – Using an implementation of the minimax algorithm with alpha-beta pruning, along with a simple evaluation function for the leaf nodes of the search tree, the engine should be able to beat most low-intermediate users. Creating an interface between engine Board class and the React app board created in Test 1, should allow for the user to play against the engine. Video evidence of more than one user playing against the engine should be provided.

Test 5 – As with any web application which contains a login system, it's important that the authentication is kept secure. For checking the passwords, the passwords should be stored using only the password's hash along with a salt to prevent rainbow table attacks. Also, I have chosen to use a token-based system for authentication to speed up API requests and allow the sharing of signed tokens. The tokens should be signed using the Elliptic Curve Digital Signature Algorithm on the curve secp256r1 up to the standard created by the National Institute of Standard and Technology. This is important as we are dealing with protected user data, so the signing algorithm should be tested using agreed values using a digital elliptic curve calculator. Evidence of this should be given using console outputs.

Test 6 – The Login and Signup form should be created to allow the user to be able to signup or login to the web app without any misunderstandings and provide useful and clear errors to the user where data inputted is wrong, missing or otherwise invalid. This can be verified using video evidence and other users' interactions with the system can be seen in later testing.

Test 7 – To allow the web app to continue, a strong API should be provided to allow the web app to access resources pertaining to the user to correctly render information on the page. The API should use correct HTTP response codes which should be tested, and data provided must be accurate protected behind the token authorisation system. This should be tested using an API querying tool, such as Postman which allows me to create requests to send to the server directly. The server console should also be visible on screen to catch any errors and show each request being sent to the server.

Test 8 – The custom game and review functionality should provide the user with the customisation settings programmed in engine. The user should also be able to share the game to users which aren't signed up to the web app. This can be verified using a mixture of screenshots and video tests.

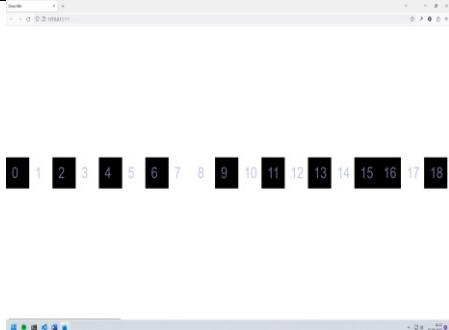
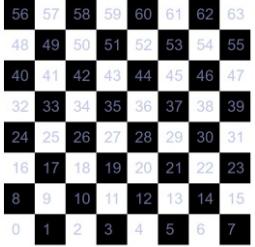
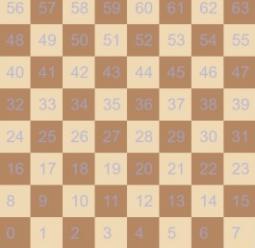
Test 9 – The adventure mode should be tested in full to make sure the difficulties are appropriate, and all text is displayed clearly and correctly. I also want to test the user's 'round up' which they get once they finish the game which should be downloadable from a HTML canvas element to a png file. This should be verified using a mixture of screenshots and a video of the entire adventure mode playthrough.

Test 10 – Final testing of the system should be conducted, testing all possible systems at once in one continuous testing video. Testing of the name change, and account deletion systems should also be tested for the first time. It would also be useful for a full end-to-end test by one other user to check the system is easy to use. This should be recorded too.

Overall, the system should be tested by at least 3 people (including myself), with all players' skill in my target user range (low to intermediate).

Test 1 - User interface for the board

Tests for the correct setting up of a HTML chessboard with programmatically placeable pieces on each square and drag-and-drop behaviour for each piece.

Test Number	Description of test	Expected Result	Actual Result	Fixes Required
1a(i)	Create the chess board that the user can see and display it correctly for the user	A grid of a chessboard with numbers on each square (for development)		Set a CSS grid for each square of the chess board 8 by 8 instead of flexbox.
1a(ii)	Create the chess board that the user can see and display it correctly for the user	A grid of a chessboard with numbers on each square (for development)		Black and white squares should be switched, and the colours should be adjusted for viewing ease.
1a(iii)	Create the chess board that the user can see and display it correctly for the user	A grid of a chessboard with numbers on each square (for development)		None
1b(i)	Display Test Position 1 (see in next section) on the board	Test Position 1 with numbers on each square		Translated the pieces by the negative of its row squares, instead of the positive of its row squares (as pieces start in the top left corner)

1b(ii)	Display Test Position 1 (see in next section) on the board	Test Position 1 with numbers on each square		Iterated the wrong direction through the list of squares
1b(iii)	Display Test Position 1 (see in next section) on the board	Test Position 1 with numbers on each square		None
1c(i)	Piece dragging behaviour	Be able to click and hold a piece to drag it		<i>Piece was not on the cursor; it was off by about a half square instead of a whole square</i>
1c(ii)	Piece dragging behaviour	Be able to click and hold a piece to drag it		<i>Piece was correctly under the cursor.</i>
1d(i)	Piece dropping behaviour standard input	Drag and drop the pawn on e2 to e4		None
1d(ii)	Piece dropping behaviour erroneous input	Drag and drop the piece to outside the board should return the piece to its original square		<i>Piece returned to its original square when dropped outside of the board</i>

Test 2 - Attack Generation

Generate the attack patterns for each piece irrespective of the placement of pieces or state of the current board. We store the attack patterns in the **Bitboard** data structure, so for each test the test output should contain a binary representation of the bitboard which represents the board when formatted 8x8.

Test Number	Description of test	Expected Result	Actual Result	Fixes Required
2a(i)	Get knight attacks on an empty board on square 39 (normal data)	A bitboard binary representation with ones on the squares 54, 45, 29, and 22 only	<p>Numerical value 2216203387392n</p> <p>Binary representation:</p> <pre>00000000 00000000 01000000 00100000 00000000 00100000 01000000 00000000</pre> <p>» </p>	Bitboard should be mirrored from the user displayed board as the function has done the moves from square 24 not 39
2a(ii)	Get knight attacks on an empty board on square 39 (normal data)	A bitboard binary representation with ones on the squares 54, 45, 29, and 22 only	<p>Numerical value 18049583422636032n</p> <p>Binary representation:</p> <pre>00000000 00000010 00000100 00000000 00000100 00000010 00000000 00000000</pre> <p>»</p>	None
2b(i)	Get knight attacks on an empty board on square 0 (boundary data)	A bitboard binary representation with ones on the squares 17 and 10 only	<p>Numerical value 132096n</p> <p>Binary representation:</p> <pre>00000000 00000000 00000000 00000000 00000000 01000000 00100000 00000000</pre> <p>»</p>	None
2c(i)	Get knight attacks on an empty board on square 64 (erroneous data)	A suitable error logged to the console	<p>! Uncaught Error using precomputed knight data</p> <p>»</p>	None
2d(i)	Get sliding piece attacks on an empty board on square 36 (rook offsets)	1s on the 5 th rank and on the e file, with zeros on all other squares	<p>GET http://127.0.0.1:5173/favicon.ico</p> <p>Numerical value 1157443723186933776n</p> <p>Binary representation:</p> <pre>00001000 00001000 00001000 11110111 00001000 00001000 00001000 00001000</pre> <p>»</p>	None

2e(i)	Get sliding piece attacks on an empty board on square 0 (bishop offsets)	1s on the diagonal from the bottom left to top right, with zeros else	<pre>Numerical value 9241421688590303744n Binary representation: 00000001 00000010 00000100 00001000 00100000 00100000 01000000 00000000</pre> <p>»</p>	None
2f(i)	Get white pawn attacks on square 36 (normal data)	1s on square 43 and 4, with zeros on all other squares	<pre>Numerical value 43980465111040n Binary representation: 00000000 00000000 00010100 00000000 00000000 00000000 00000000 00000000</pre> <p>» </p>	None
2g(i)	Get black pawn attacks on edge square 24 (boundary data)	1 on 17 only, all others should be 0	<pre>Numerical value 131072n Binary representation: 00000000 00000000 00000000 00000000 00000000 01000000 00000000 00000000</pre> <p>» </p>	None
2h(i)	Get black pawn attacks on last file, square 0 (erroneous data)	A suitable error thrown in the console	<pre>Numerical value 0n Binary representation: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000</pre> <p>» </p>	Added a check for the pawn being on the last file, throwing an error if it is.
2h(ii)	Get black pawn attacks on last file, square 0 (erroneous data)	A suitable error thrown in the console	<p>visit https://reactjs.org/link/error-boundaries to learn more</p> <p>⚠️ Uncaught Board state is invalid, a pawn is on an end rank!</p> <p>»</p>	None
2i(i)	Get king attacks on square 0	1s on the squares 1, 8 and 9, with zeros on all other squares.	<pre>[vite] connected. Numerical value 770n Binary representation: 00000000 00000000 00000000 00000000 00000000 00000000 11000000 01000000</pre>	None

Test 3 - Move Generation

PERFT tests are tests where the engine analyses the number of moves in a position and plays each move on the board. For each move played, that board is then PERFT tested recursively until a maximum depth is reached. The test then outputs the number of nodes reached (positions where the final depth has been reached). All tests are done on the same board in memory, playing and unplaying moves, so we also test the delicate play move and unplay move methods. Any major errors missed in this test will cause the engine to be obviously misevaluating positions and playing very poorly.

PERFT test positions and community agreed results at https://www.chessprogramming.org/Perft_Results

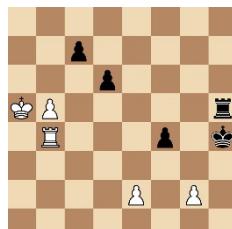
Test Position 1



Test Position 2



Test Position 3



Test Position 4



Test Position 1 was chosen as it's the starting position for a chess game, and its simplicity made it a good candidate for the first test position, as I ran into each bug one at a time, whereas I might have got overloaded with bugs if I used a more complex position.

Test Position 2 was chosen as both sides can promote, promote with capture, black can castle both sides, and en passant is possible. These moves are the hardest to code so I could uncover many bugs which you wouldn't see in most positions. I also mirrored the board for in Test m, to test white's short and long castling functionality.

Test Position 3 was chosen as the number of pieces is much smaller, so I could go more moves deeper into the analysis which would test the unplay move function further.

Test Position 4 was chosen as it caused disagreements in early 2000s about the correct PERFT results and tested castling further than **Test Position 2**.

Test Number	Test Description	Expected Result	Actual Result	Fixed Required
3a(i)	Calculate number of moves from position with depth 1 in Test Position 1	20 Nodes	20 Nodes	None
3b(i)	Calculate number of moves from position with depth 2 in Test Position 1	400 Nodes	400 Nodes	None
3c(i)	Calculate number of moves from position with depth 3 in Test Position 1	8902 Nodes	8902 Nodes	None
3d(i)	Calculate number of moves from position with depth 4 in Test Position 1	1972781 Nodes	1924305 Nodes	The value of captured pieces was not being saved to the board's past game state stack, so captures were not being unplayed correctly. The rank that a pawn needed to be on to

				be able to capture and promote was set to 6. (It should have been 6 when white and 1 when black.)
3d(ii)	Calculate number of moves from position with depth 4 in Test Position 1	1972781 Nodes	1972781 Nodes	None
3e(i)	Calculate number of moves from position with depth 5 in Test Position 1	4865609 Nodes	4865609 Nodes	None
3f(i)	Calculate number of moves from position with depth 6 in Test Position 1	119060324 Nodes	1190603 24 Nodes	None
3g(i)	Calculate number of moves from position with depth 1 in Test Position 2	6 Nodes	6 Nodes	None
3h(i)	Calculate number of moves from position with depth 2 in Test Position 2	264 Nodes	258 Nodes	Accidental overwriting of pieces between the king and the rook in castling
3h(ii)	Calculate number of moves from position with depth 2 in Test Position 2	264 Nodes	264 Nodes	None
3i(i)	Calculate number of moves from position with depth 3 in Test Position 2	9467 Nodes	9461 Nodes	Unable to promote to a knight, instead the option to promote to a rook was repeated
3i(ii)	Calculate number of moves from position with depth 3 in Test Position 2	9467 Nodes	9467 Nodes	None
3j(i)	Calculate number of moves from position with depth 4 in Test Position 2	422333 Nodes	422333 Nodes	None
3k(i)	Calculate number of moves from position with depth 5 in Test Position 2	15833292 Nodes	1583415 2 Nodes	Queen-side castling could occur even if there was a piece on b1/b8
3k(ii)	Calculate number of moves from position with depth 5 in Test Position 2	15833292 Nodes	1583329 2 Nodes	None
3l(i)	Calculate number of moves from position with depth 6 in Test Position 2	706045033 Nodes	7060450 33 Nodes	None

3m(i)	All tests from 1.g to 1.l repeated, but with the board in Test Position 2 but flipped with white as black and black as white (mirrored)	6 Nodes, 264 Nodes, 9467 Nodes, 422333 Nodes	6 Nodes, 264 Nodes, 9467 Nodes, 422333 Nodes	None
3n(i)	Calculate number of moves from position with depth 1-7 in Test Position 3	14 Nodes, 191 Nodes, 2812 Nodes, 43238 Nodes, 674624 Nodes, 11030083 Nodes 178633661 Nodes	14 Nodes, 191 Nodes, 2812 Nodes, 43238 Nodes, 674624 Nodes, 1103008 3Nodes 1786336 61 Nodes	None
3o(i)	Calculate number of moves from position with depth 1-5 in Test Position 4	44 Nodes, 1486 Nodes, 62379 Nodes, 2103487 Nodes, 89941194 Nodes	44 Nodes, 1486 Nodes, 62379 Nodes, 2103487 Nodes, 8994119 4 Nodes	None

All tests from **Test 3** (excluding 3G-3L) are setup to run automatically and I retested the program whenever I made any changes to the engine.

Code for the automatic testing is in *frontend/src/routes/Test*.

Fully automatic testing (1 depth below max depth tested in Test 3)

Link to video: <https://www.youtube.com/watch?v=o8PPSLyWkxk>

Test 4 - Engine

Tests here are done on the Search function, the Evaluation function, and the Engine class which acts as an interface between the board the user sees and the rest of the Engine.

Test User 1 – Me

Test User 2 – The user in Test 4

Test User 3 – The user in Test 10

Test Number	Description of test	Expected Result	Actual Result	Fixes Required
4a(i)	Zero depth evaluation of Test Position 1	0 centipawns	0 centipawns	None
4b(i)	Zero depth evaluation of Test Position 2 (mirrored)	A positive value (white is winning)	105 centipawns	None
4c(i)	Depth 3 Engine against Test User 1	The engine beats Test User 1	Check video link. The engine played very strangely, loosing pieces and lost the game, but not playing randomly or purposely bad	The alpha and beta evaluations should have been recursively given as the negative of the current alpha or beta (as a position good for white is bad for black).
4c(ii)	Depth 3 Engine against Test User 1	The engine beats Test User 1	Check video link. The engine played much better, eventually getting checkmate, but Test User 1 drew the game by repetition which the engine did not call.	A check for draws by repetition was added, by storing all past positions and checking if any had come up before when analysing a new position.
4d(i)	Depth 3 Engine against Test User 2	The engine beats Test User 2	Check video link and Transcription 1 as the room was quite loud and some of the conversation might be hard to hear. The engine beat Test User 2 as expected. Test User 2 also made some comments on the design (see Transcription 1)	Added a colour indicator when to where the engine just moved as Test User 2 struggled to see the engine checking him. Also pushed the computer's calculation function below the render function, so the board wouldn't freeze, as Test User 2 described.

Test 5 - Cryptographic functions

These tests are to make sure both the password hashing and salting is done correctly, and the token signing is working and secure.

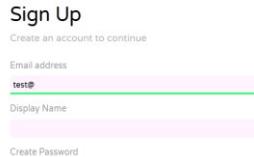
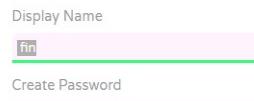
Expected results for the elliptic curve calculations were calculated on the website <http://christelbach.com/ECCalculator.aspx> with the curve parameters of secp256r1.

Test Number	Test Description	Expected Result	Actual Result	Fixes Required
5a(i)	Generate a salted password hash and recheck the password hash to verify the password	The program outputs a long hex string, followed by an integer and then 'True' when the hashes match	Screenshot 1	None
5b(i)	Double the generator point on the elliptic curve	Logging of the point: (565152197906911714131090...6242040, 337703184371225825922371...7583569)	Screenshot 2	I was using normal division, not modular division, so my calculations were being done with floating-point numbers.
5b(ii)	Double the generator point on the elliptic curve	Same as above	Screenshot 3	None
5c(i)	Compute 7592346587435283943G on the elliptic curve (arbitrary scalar)	Logging of the point: (456575617249268022440549...0027541, 107163266788817127308136...649759)	Screenshot 4	None
5d(i)	Sign a random string of bytes using and private key and verify the signature using the public key.	A very long integer being logged as the signature, and the logging of 'True' when the signature is verified.	Screenshot 5	None
5e(i)	Sign a Python dictionary and output a base64 encoded JSON message with the signature attached.	A base64 encode string starting with 'ey' (<i>the base64 encoding of the '{' symbol</i>) and a logged python dictionary with the message 'failure' of False	Screenshot 6	The differences between spaces between each attribute when encoding the dictionary into JSON, changed the message slightly.
5e(ii)	Sign a Python dictionary and output a base64 encoded JSON message with the signature attached.	A base64 encode string starting with 'ey' and a logged python dictionary with the message 'failure' of False	Screenshot 7	None

Test 6 - Login and Signup form

These tests are to test that the login and signup form cannot send invalid information to the server. Tests 6a-6m are on the signup form, and tests 6n-6q are on the login form. A video of all tests passing is in [Testing video links](#).

For the purposes of the tests, the password input type was removed, so the value of the password input box can be viewed by the examiner.

Test Number	Test Description	Expected Result	Actual Result	Fixes Required
6a(i)	Entering an invalid email “test@”	The email box goes red and displays the error “Please enter a valid email”		The email regex did not work on Firefox, so I found a new Regex to use.
6a(ii)	Entering an invalid email “test@”	The email box goes red and display the error “Please enter a valid email”	Check video link. The correct error was shown.	None.
6b(i)	Entering a valid email “test@site.com”	The email box goes green.	Check video link. The box accepted the email.	None.
6c(i)	Entering the too short display name “a”	The display name box goes red and display the error “Names must be between 2 and 32 characters and not end or start with a space”	Check video link. The box correctly rejected the name.	None.
6d(i)	Entering a name that starts with a space “ name”	The display name box goes red and displays the error “Names must be between 2 and 32 characters and not end or start with a space”		Changed the regex to disallow spaces at the start of the names.
6d(ii)	Entering a name that starts with a space “ name”	The display name box goes red and displays the error “Names must be between 2 and 32 characters and not end or start with a space”	Check video link. The box correctly rejected the name.	None.
6e(i)	Entering a valid name “Finley”	The display name box goes green and accepts the name.	Check video link. The box correctly accepted the name.	None.
6f(i)	Entering the valid password “Password1”	The password box goes green and accepts the password	Check video link. The box correctly accepted the password.	None.
6g(i)	Entering the invalid password “Password” (no number)	The password box goes red and displays the error “Passwords must contain between 8 and 32 characters, at least 1 uppercase, 1 lowercase, and 1 number”	Check video link. The box correctly rejected the password.	None.
6h(i)	Entering the valid password “Password1”	The confirm password box goes red and displays the	Check video link. The box correctly	None.

	and the invalid confirm password of “Password1a”	error “Passwords do not match”	rejected the password.	
6i(i)	Entering the valid password “Password1” in both the password and confirm password boxes	Both boxes go green and accept the password	Create Password Confirm Password Passwords do not match	I was comparing the wrong variables. Also, the green was shown under the confirm password input, so I fixed that.
6j(i)	Entering the valid password “Password1” in both the password and confirm password boxes	Both boxes go green and accept the password	Check video link. The box accepted the password.	None.
6k(i)	Submit with empty display name input	The display name box goes red and displayed the error “This field must not be left blank”. The form is not submitted	Check video link. The form is not submitted.	None.
6l(i)	Submit with an invalid email	No change (the error should already be showing). The form is not submitted.	Check video link. The form is correctly not submitted.	None.
6m(i)	Submit with an email with an account which already exists	The form is not submitted, and the user is given an error message showing them an account already exists with that email.	Check video link. The form is correctly not submitted, and an error is shown.	None.
6n(i)	Submit with all details valid.	The form is submitted, and the user is redirected to the login form	Check video link. The form is submitted, and the user is redirected correctly.	None.
6o(i)	Login with the incorrect details from the previous tests	The boxes go red and the error “The credentials provided were invalid” is displayed	Check video link. The correct error is returned	None.
6p(i)	Login with the correct email but incorrect password.	The boxes go red and the error “The credentials provided were invalid” is displayed	Check video link. The correct error is returned	None.
6q(i)	Login with all correct details	The user is redirected with a cookie for authentication and redirected to the root page. The cookie is correctly signed and contains the user’s id.	Check video link. The user gets the cookie and is correctly redirected.	None.

Test 7 - API and Authorisation

These tests are on the REST API which acts as an interface between the web app and the database. It also handles logic for determining which users can access what data. Testing here is done using Postman to construct the web requests and analyse the responses. This will also test the SQL database queries. I had to manually remove the ‘Secure’ labels on the token cookies, as with this label on the cookie wouldn’t be sent over HTTP, which is what I was using for testing locally.

All the tests ran correctly which I was expecting, since almost all these functionalities had been tested before in **Test 6**, however while making the tests, I did find one error, where providing a single missing attribute with a non-empty request body would cause an error, which I fixed before recording the tests. The tests were recorded (go to **Testing Video Links**). At the start of the video, I delete the SQLite database file, so we’re testing from a fresh database, and I also have a terminal on screen which logs all the requests and responses from the server.

Test Number	Test Description	JSON Response
7a	Signup with invalid email	Valid email not provided in the request
7b	Signup with invalid name (trailing space)	Valid display name not provided in the request
7c	Signup with invalid password (too long)	Valid password not provided in the request
7d	Signup with empty request body	No data was provided in the request
7e	Signup normally	Account created - Please Login
7f	Login with incorrect password	The credentials provided were invalid
7g	Login normally	Successfully logged in (<i>token and isLoggedIn cookie should also be sent in the response Set-Cookie header</i>).
7h	Get user (self with @me)	User data
7i	Get user with user id	Same as above
7j	Get a different user	Forbidden (<i>even if the user doesn't exist</i>)
7k	Store a chess game under our user id	Game successfully archived
7l	Same as above but without a ‘winner’ attribute set in the request body	All game data not provided in the request
7m	Store a chess game under a different user id	Forbidden
7n	Get all games from self	Data sent from test 7k
7o	Get game from 7k by its id	Same as above
7p	Get game with invalid id	Game does not exist
7q	Get text from adventure level (no cookie)	Adventure level data
7s	Change user adventure level to 2.	User level id updated (<i>check by conducting test 7i again</i>)
7t	Change user display name	User display name updated (<i>check by conducting test 7i again</i>)
7u	Create sharable link from game in test 7k	A link path starting /s/ is given.
7v	Delete account	Cookies are revoked and new account can be made by conducting test 7e again
7w	Follow shareable link (while not logged in) to get a game scoped token. Conduct test 7o again to verify the token works	HTML response and tempToken is sent. The game is sent in response to test 7o
7x	Follow invalid shareable link	Link does not exist (<i>Plain text not JSON</i>)
7y	Sign in with admin account	Successfully logged in.
7z	Get all users (<i>only works with admin</i>)	Array of all users on the database.

Test 8 - Custom Game and Review

This is testing for the custom game feature and the review functionality. This is at the core of the project: the vast customisation settings for the engine are what differentiates my system from the existing systems. It is also important that the difficulty setting is wide enough to allow a range of users to be able to use the system, otherwise users could find themselves locked out of the adventure mode, not being able to pass the level.

Test Number	Test Description	Expected Result	Actual Result	Fixes Required
8a(i)	Make sure the values of the sliders are recorded in the engine. (Max on all sliders)	Engine depth = 4 Position Strength = 100 Aggressiveness = 100 Blind Spots = 50 Piece Exchanging Tendency = 100	Screenshot 8 <i>(all correct apart from blind spots which is incorrectly 100)</i>	Changed limit on blind spot slider to 100.
8a(ii)	Same as above	Same as above	Screenshot 9 <i>(all correct)</i>	None.
8b(i)	Test the difference between depth at 1 and depth at 4.	Depth at 1 should make moves almost instantly and depth at 4 should take up to 30 seconds per move.	Check video link (0:00-1:25) <i>As expected</i>	None.
8c(i)	Test the difference between positional strength 0 and 100.	With the positional strength 0, the pieces will be spread out randomly, with positional strength 100, pieces will tend towards the opponent's side in the middle of the board.	Check video link (1:25-3:00) <i>As expected</i>	None.
8d(i)	Test the difference between aggressiveness 0 and 100.	The engine should keep its pieces on its own side at low aggressiveness and throw its pieces at the opponent at high aggressiveness.	Check video link (3:00-5:05) <i>As expected</i>	None.
8e(i)	Test the difference between blind spots from 0 to 50.	The engine should play normally at 0, and extremely poor at 50.	Check video link (5:05-7:30) <i>As expected</i>	None.
8f(i)	Test the saving and archiving of a game, and the review of the game by stepping through move-by-move.	While testing, a game should be finished, saved, and then rewatched, with the save moves shown.	Check video link (7:30-8:34) <i>As expected (ignore the other games on the history from past tests)</i>	None.
8g(i)	Share the link from the game in 8f and view the game while not logged in.	To be able to view the game the same as in 8f after being redirected from the /s/ link.	Check video link (8:34-9:13) <i>As expected</i>	None.
8h(i)	Same as 8g but logged into a different account not normally available to access the game. (Check the temp token overrides the normal token.)	Same as above, and not be able to view the game when the 'tempToken' cookie is removed, getting the Forbidden 403 HTTP status code.	Check video link (9:13-10:14) <i>As expected</i>	None.

Test 9 - Adventure Mode

This is testing for the main feature of the project, the adventure mode. It's important that the adventure mode can be beaten by most chess players, and that there are no bugs that prevent the user from advancing, locking them out of the main functionality of the project. We will also test the round up analysis when the user finishes the adventure mode, and test the updating and exporting of the feature, as this is also one of the main differences that this project has over existing online chess sites.

Test Number	Test Description	Expected Result	Actual Result	Fixes Required
9a(i)	Test the positioning and size of the story text and continue button	Text centred in the middle-top of the page, large enough to easily read. Button should be lower than the middle vertically.	Screenshot 10 <i>(Text is too low and small)</i>	I moved the text up and increased the font size.
9a(ii)	Same as above	Same as above	Check video link <i>(all correct)</i>	None.
9b(i)	Test the transition from the story to the chess game	When the user finishes the chapter, they should press continue which should put them in a chess game.	Check video link <i>(all correct)</i>	None.
9c(i)	Test the actual game (for the first chapter)	The first battle should be very easy to beat, which the engine throwing its pieces towards the player aimlessly.	Check video link. <i>(The engine played too well, so I had to stop the test here to see what went wrong)</i>	Increased depth from 1 to 2. This meant that the simplify position evaluation is done with a depth of 3, so more moves are missed out.
9c(ii)	Same as above	Same as above	Check video link <i>(all worked fine)</i>	None.
9d(i)	Test speech text.	Speech should be prefixed with the coloured speaker name	Check video link <i>(all worked fine)</i>	None.
9e(i)	Full test of the adventure.	All speech should be displayed, and all games played suitably easy for a low skilled player.	Check video link <i>(all worked fine, more difficulty testing will be in Test 10)</i>	I wasn't happy with how some opponents played, but I'll wait for my testers in Test 10 before I change anything.
9f(i)	Test the game round up feature. Lose a game and recheck.	Get the win-ratio of 100% for all sections. Then lose the game, the number of games played, and games lost increases by one, the aggressive win rate changes to 75%, and the tactically strong win rate to 75%.	Check video link <i>(all worked fine)</i>	None.
9g(i)	Test the download for the round up.	The same round up is saved as a .png file to the user's device.	Check video link <i>(all worked fine)</i>	None.

Test 10 - End-to-End testing

Tests of the entire system from signup to finishing the account. All the tests here are tests repeated from earlier tests, I'm only testing how the components of the system fit together and some of the miscellaneous components which haven't been tested so far. I also am conduction end-to-end testing by other people who haven't used the system before. This will be helpful to judge how easy and intuitive the app is to use, which is important, as my expected users are on the younger side, so might need more assistance navigating the platform that I thought. If user requests or requires a major change to fix a problem, I will include this in my evaluation instead, as I am nearing the end of my project.

Test 10.1 - My End-to-End Test Action Log

Full video link in [Testing Video Links](#)

Action	Timestamp	Errors, unexpected behaviour, or user confusion	Fixes Required
Redirected from root path to the logged in path.	0:00	None.	None.
Clicked link to signup	0:03	None.	None.
Entered valid email address	0:09	None.	None.
Entered invalid name	0:14	None.	None.
Entered valid name	0:16	None.	None.
Entered invalid email	0:20	None.	None.
Entered valid password	0:28	None.	None.
Entered invalid confirm password	0:32	None.	None.
Entered valid confirm password	0:33	None.	None.
Completed sign up	0:35	None.	None.
Logged in with valid credentials	0:45	None.	None.
Started custom game as white	1:10	None.	None.
Played custom game to a stalemate.	3:25	None.	None.
Viewed game history	3:37	Page incorrceted showed 'Loss by Stalemate'.	Replaced 'Loss' with 'Draw' in the table when winner is 0 (no winner).
Reviewed while logged in.	3:46	None.	None.
Created shareable link	4:27	None.	None.
Reviewed from shareable link in private window (not logged in)	4:30	None.	None.
Reviewed from non-shareable link in private window (not logged in)	5:02	The API rejected the request as designed, but the user wasn't notified that they can't access the game.	Redirected users to the login page if the server sent 404 (game not found), 401 (unauthorised), or 403 (forbidden).
Started the adventure mode	5:20	None.	None.
Display name 'Finley' formatted in dialogue.	6:35	None.	None.
Left adventure mode mid-story to change display name to 'James'.	6:40	None.	None.

Adventure mode continues from last saved chapter, with the new display name ‘James’	6:55	None.	None.
Finish adventure mode and viewed statistics sheet.	15:08	None.	None.
Download and view locally statistics sheet.	15:15	None.	None.
Viewed game history	15:31	Game result showed as ‘Win by Checkmate’	Changed to ‘Win by Checkmate’
Reviewed most recent game	15:34	The next move button was positioned incorrectly and way too big.	Changed conflicting CSS class names of the board container with the Adventure board container.
Reviewed next most recent game	15:50	Same as above	Same as above.
Deleted account	16:00	None.	None.
Logged in with server-generated admin credentials.	16:07	None.	None.
Reviewed empty game history	16:12	None.	None.
Navigated to /api/users/all/	16:22	I mistyped the route while testing. Ignore the NGINX error.	None.
Navigated to /api/users/all to check the user was deleted.	16:53	None.	None.

Test 10.2 – Test User 3 End-to-End Test Action Log

Full video link in [Testing Video Links](#)

This is also a test of the project being hosted on Google Cloud, as before tests were conducted either locally or through static file hosting (where the API was not included).

Action	Timestamp	Errors, unexpected behaviour, or user confusion	Fixes Required
User navigates to the URL given and is redirected	0:01	None.	None.
Submits an email and password in the login form	0:19	The system acted as expected, but the test user thought the login form was the signup form. There’s no easy fix for this, which I talk about more in the evaluation section.	None.
User navigates to the sign-up page	0:25	None.	None.
User enters an email address using Edge autofill	0:32	Microsoft Edge auto-filled an email which the user inputted as the login credentials; however, Edge also filled the email address for the display name box.	Added an ‘autocomplete=”off”’ HTML attribute to the name input.
User enters the unsecure password “password”	0:37	None.	None.

User enters the confirm password input incorrectly	0:55	None.	None.
User changes display name	1:09	None.	None.
User enters matching password	1:16	None.	None.
User signs up	1:18	None.	None.
User logs in	1:28	None.	None.
User clicks on 'Settings'	1:41	None.	None.
User clicks on custom game	1:51	None.	None.
User creates an aggressive engine, with high blind spots	2:10	None.	None.
User starts the game	2:11	The board was too zoomed in, which resulted in the board being only partially visible. On my objectives, I put aside making the website responsive, so this is outside of the scope of the project.	None.
User plays the game	2:12-5:26	The engine plays about expected, with a good amount of aggressiveness, and played down to the lower skill level tester well.	None.
The user wins the game by checkmate	5:26	None.	None.
The user plays the first level of the 'adventure' mode.	5:38	The test user played much worse than I anticipated here, so it's very reassuring that they were still able to beat the engine on the engine's lowest settings.	None.
The user clicks on game history page.	9:15	None.	None.
The user reviews the game in the adventure mode.	9:22	Same error with the incorrectly positioned button, as I was accidentally running an older version of the code. The user also wished for a 'back' button for being back a move.	None (apart from updating the project version in the cloud).

Post-test Interview:

Me: "Do you have any immediate problems with the system which make it hard to use or navigate"

Test User 3: "There was nothing that made it hard to navigate, but there wasn't exactly links between each section so you [don't] have to step back to get to the main page."

Me: "Did the engine play human-like and play to a suitable level which you set"

Test User 3: "Well, I believe that with sliders and settings it has the potential to act human like, but with blind spots set to 50% it doesn't play human like 50% of the time. Is your intention to make it play human-like?"

Me: "Yes, but I want to make sure that everyone can still beat the engine by changing the settings."

Test User 3: "Hmm. It just doesn't play human like as much in the extremes."

Screenshots

Screenshot 1:

```

83
84     combined_numeric = salt & (password_numeric << (constants.salt_bytelength * 8))
85
86     combined_bytes_length = math.ceil(((math.floor(math.log2(password_numeric)) + 1) / 8) + constants.salt_bytelength)
87     combined_bytes = int.to_bytes(combined_numeric, combined_bytes_length)
88
89     return sha256(combined_bytes, usedforsecurity=True).
90
91
92     def check_password_hash(password: str, password_hash: str):
93         digest_hex, salt_string = password_hash.split(" ")
94
95         check_digest_hash, _ = create_password_hash(password)
96
97         return check_digest_hash == digest_hex
98
99
100    if __name__ == "__main__":
101        a = create_password_hash("password")
102        print(a)
103
104        print(check_password_hash("password", a))
105

```

Git Bash

```

finle@FinleysComputer ~/Coding/newest-nea/nea/backend/crypto_auth (main)
$ python __init__.py
15ec7bf0b50732b49f8228e07d24365338f9e3ab994b00af08e5a3bffe55fd8b 435417899
True

```

Git Bash

```

finle@FinleysComputer ~/Coding/newest-nea/nea/backend/crypto_auth (main)
$ |

```

Screenshot 2:

```

194     G=(
195         484395612939064517590525852527979142027629495260417479958440807170824046351
196         361342509567497957985851279195878819566111066729850150718771982535684144051
197     ),
198     n=11579208921035624876269744694940757352999695522413576034242225906106851204436
199 )
200
201 if __name__ == "__main__":
202     new_point = curve.point_addition(curve.G, curve.G)
203     print(new_point)
204

```

Git Bash

```

finle@FinleysComputer ~/Coding/newest-nea/nea/backend/crypto_auth (main)
$ python elliptic_curve.py
(3.2952382864281573e+76, 4.997046695979657e+76)

```

Git Bash

```

finle@FinleysComputer ~/Coding/newest-nea/nea/backend/crypto_auth (main)
$ |

```

Screenshot 3:

```

199
200
201 if __name__ == "__main__":
202     new_point = curve.point_addition(curve.G, curve.G)
203     print(new_point)
204

```

Git Bash

```

finle@FinleysComputer ~/Coding/newest-nea/nea/backend/crypto_auth (main)
$ python elliptic_curve.py
(56515219790691171413109057904011688695424810155802929973526481321309856242040, 3
377031843712258259223711451491452598088675519751548567112458094635497583569)

```

Git Bash

```

finle@FinleysComputer ~/Coding/newest-nea/nea/backend/crypto_auth (main)
$ |

```

Screenshot 4:

```

200
201     if __name__ == "__main__":
202         new_point = curve.scalar_multiplication(7592346587435283943, curve.G)
203         print(new_point)
204
finle@FinleysComputer ~/Coding/newest-nea/nea/backend/crypto_auth (main)
$ python elliptic_curve.py
(45657561724926802244054908658540479513960907182086622596357808504780027541447, 1
07163266788817127308136553397263134791629950148989282633809352972864975974054)

```

Screenshot 5:

```

195     48439561293906451/59052585252/9/914202/62949526041/4/995844080/1/082404635286,
196     36134250956749795798585127919587881956611106672985015071877198253568414405109,
197     ),
198     n=115792089210356248762697446949407573529996955224135760342422259061068512044369,
199   )
200
201     pub_key = 991771802931504373747901948739580884575920666552168810380409399328466584793005227594086975
202     priv_key = 50011543548814054331552347745764341220005386584648787047068252027433500990510
203     if __name__ == "__main__":
204         random_bytes = b"\xa5\xfe\x43"
205
206         signature = curve.createSignature(random_bytes, private_key=priv_key)
207         print(signature)
208         print(curve.verifySignature(random_bytes, signature=signature, public_key_int=pub_key))
209
finle@FinleysComputer ~/Coding/newest-nea/nea/backend/crypto_auth (main)
$ python elliptic_curve.py
8334517439314271441335864943560041193212878799033210132349998077859109277928629336629
352502938105868860106648149701336214349450103865710930241257292099979
True

```

Screenshot 6:

```

98
99     priv_key = 1507053737913644028850512931965257038898911731619150639023704795659490118767
100    public_key = 40744359496229070217131348263833368977996742127029360704566521063031521960489540178
101    if __name__ == "__main__":
102        token = create_token({"test_key": "test_value"}, private_key=priv_key)
103        print(token)
104
105        decoded_token = verify(token, public_key=public_key)
106        print(decoded_token)
107
finle@FinleysComputer ~/Coding/newest-nea/nea/backend/crypto_auth (main)
$ python __init__.py
eyJpbnZhbGlkQXQiOjE2OTU0TY1MjQuOTc10Tc5LCJzaWduYXR1cmUiOiJ0SGJVend6aWd3cFNRejhNdkEwW
TcxUVFCbW8rNXZQalc3dGJRNxDwcXIzUWR0cW42R21Sbkc1S3AvbzN2d2tSaFBGdkdYQ1B6TDI0T1NsbHEvb0
htUT09Iiwic2lnbmVkQXQiOjE2OTU0MTAxMjQuOTc10Tc5LCJ0ZXN0X2tleSI6InRlc3RfdmFsdWUifQ==
{'failure': 'Token signature invalid'}
finle@FinleysComputer ~/Coding/newest-nea/nea/backend/crypto_auth (main)

```

Screenshot 7:

```

97
98
99     priv_key = 15070537379136440288505129319652570388989117316191506394023704795659490118767
100    public_key = 407443594962290702171313482638333689779967421270293607045665210630315219604895401
101    if __name__ == "__main__":
102        token = create_token({"test_key": "test_value"}, private_key=priv_key)
103        print(token)
104
105        decoded_token = verify(token, public_key=public_key)
106        print(decoded_token)
107

```

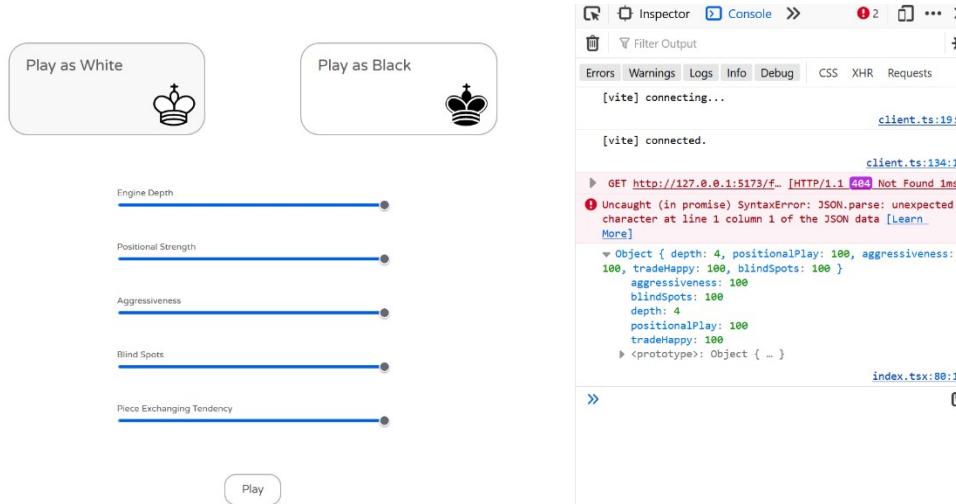
```

finle@FinleysComputer ~/Coding/newest-nea/nea/backend/crypto_auth (main)
$ python __init__.py
eyJpbnZhbGlkQXQiOjE2OTU0OTY2MTkuMDcyNDA5NCwic2lnbmF0dXJlIjoiQlpYUWhBa01XNVZuRTd5TUpwU
3EzYWRVRjdGNh1hNDh4MXptWER4d25YbmZRVRVYkFmZ3BFTjZZVkJVBRVJJTDcxYZZ0WjlUc20yekFCSFg4SV
F6V2c9PSIsInNpZ251ZEf0IjoxNjk1NDEwMjE5LjA3MjQwOTQsInRlc3Rfa2V5IjoidGVzdF92YWx1ZSJ9
{'test_key': 'test_value', 'failure': False}

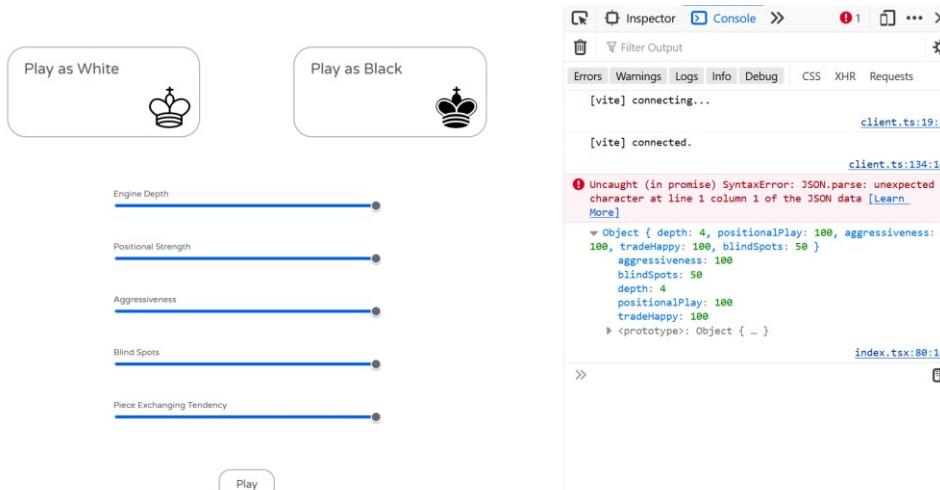
finle@FinleysComputer ~/Coding/newest-nea/nea/backend/crypto_auth (main)

```

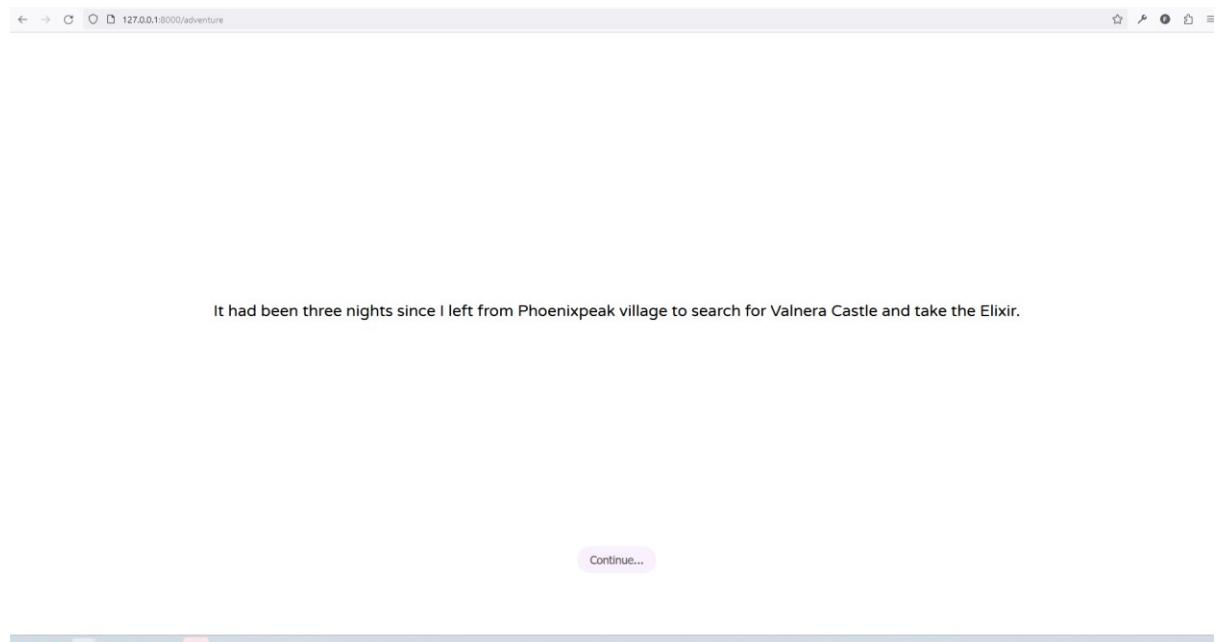
Screenshot 8:



Screenshot 9:



Screenshot 10:



Testing video links

Manually Change the quality of the YouTube video player if the text is not visible.

Link to playlist containing all videos.

Shortened URL: <https://tinyurl.com/finleynea>

Direct URL to playlist (if shortened URL does not work)

https://www.youtube.com/playlist?list=PLIBnTcOPbWiZJrxLaCyaOol_HuE9aEGd_

URLs to individual video links

Test Number	Link
Test 4c(i)	https://youtu.be/GoRMgk2nbGQ
Test 4c(ii)	https://youtu.be/vnRzjOp8XM
Test 4d(i)	https://youtu.be/fDdmlVYvz_4 Also see Transcription 1
Test 6	https://youtu.be/4OYXiSrRNRI
Test 7	https://youtu.be/NFeElG-1z7M
Test 8b to 8h	https://youtu.be/vzhvRlnU7AE
Test 9a(i) to 9c(i)	https://youtu.be/wj_GGvfbb50
Test 9c(ii) to 9g(i)	https://youtu.be/Sn6G_7zFo74
Test 10.1	https://youtu.be/xXpvwnKMawg
Test 10.2	https://youtu.be/zmDbIbeJUz0

Transcriptions

Important comments are bolded.

Transcription 1

[Me]: Right, go on.

[Me]: You can talk about it, talk about the whole system as a whole.

[Test User 2]: Ok, alright, alright, alright.

[Me]: Tell me what's bad and tell me what's good.

[Test User 2]: I'm gonna win. The bad thing is that I'm losing.

[Me]: It might not be bad thing; this is it on the high settings, so.

[Test User 2]: Oh, this is the high settings?

[Me]: Yeah, you should be expecting a hard game.

[Test User 2]: See that was intentional. (*sarcasm*)

[Test User 2]. I'm gonna bring the bishop up to defend. Then get, then that's gonna happen. But that was intentional.

[Me]: Ok

[Test User 2]: But not ready for this – ah see defence. It knows. The engine knows.

[Test User 2]: Just do some pushing. I'm in check, so I'm just gonna-. Ok I'll trade that.

[Test User 2]: **Little bit slow** but that's ok.

[Me]: Ok, ok.

[Test User 2]: That was a terrible move (*talking about his own move*)

[Test User 2]: No that was intentional (*sarcasm*). Right here me out, bring- ah bring it up. But then what I do then, hmpf.

[Test User 2]: Strategies, strategies, **I'm in check**, whoops.

[Test User 2]: See I've traded with the queen now, so it's an easy win for me.

[Me]: [The game] doesn't look too mismatched.

[Test User 2]: See it works out, in the end it will work out in my favour. Let's try-, we're not having any of that.

[Both]: Oooh (*The engine played a good move that neither of us saw*)

[Me]: It got you there.

[Test User 2]: That's fine, I'll take that and check you. Then I'll try and push

[Me]: Let's see if finishes the job.

[Test User 2]: I'm gonna quickly bait this, there we go. Bang.

[Me]: I think it got you.

[Test User 2]: No, no, no, it's definitely not got me. See?

[Me]: I wanna see if it actually checkmates you and doesn't just check you over and over again. Because that's the problem I had on the test before.

[Me]: It's making progress.

[Test User 2]: What we are gonna do, however, is move this guy up. Then, alright, move this guy up, then keep pushing.

[Me]: Uh-oh

[Test User 2]: Ok? That's quite interesting.

[Crosstalk about the engine's impressive move]

[Me]: [It's] checkmate.

[Test User 2 tries unsuccessfully to break the board by moving the king outside of the container]

[Me]: So what you do say about it, what do you say about the system? Would you say it's easy to use?

[Test User 2]: **It's easy to use. It's a tiny bit slow, but that's ok.**

[Me]: Tiny bit slow, ok. **Is it slow because it freezes when you [make a move]?**

[Test User 2]: Yeah. And it's also too good.

[Me]: Well, it's on the harder settings, so it's what I'm expecting.

4 Evaluation

Changes Due to Feedback

Throughout the project I have had other people test my project and give me feedback to improve the project. For my first test of just the engine with Test User 2, I added an indicator of where the user and engine had just moved, so the positions were more easily understood. Test User 2 also complained that the engine would freeze the board while it was thinking, causing the piece to ‘hover’ over the board until the engine had finished calculating. To fix this, I added a delay on the computer calculation function, so the rendering method on the board React component had processor time to be computed first.

From Test User 3, I only made some minor changes to the actual code, such as preventing the name input from being autocompleted into, and I added some navigational features, such configuring the app to push the routes to the browser history so they can be easily traversed by the user.

My interviewee also had some comments and first introduced me to the repetition problem while he was testing the project outside of our interview, which lead me to introduce the past board stack and the hashing function to encourage the engine to stop repeating positions, which worked as I no longer had any more problems in the subsequent tests.

Project Objective Evaluation

These are the evaluation of the objectives discussed in the analysis section, how I think I achieved them, and how my testers observed the project. (*Look back at the Analysis section for specificity for each objective*)

1. Create a signup and login system.
2. Create a customisable chess engine.
3. Create an adventure mode using the engine customisations.
4. Create a REST API between the database and the website.

Objective 1 – The signup and login system

The signup and login system’s functionality were complete, and all requirements for the first objective were met fully. The assurance of the complexity of the password was assured in Test 6, and the server-side protections were confirmed in Test 5 and 7.

Beyond the objectives, in the end-to-end test with Test User 3 the user thought that the first form they were redirected to be the sign-up form, where in fact it was the login form. If I was going to redo the project, I could add some browser side cookie, which is stored when the user first navigates to the website. The absence of this cookie would cause the user to be redirected to the sign-up page instead of the login page.

Objective 2 – The engine

The resources used by the engine on the browser have not caused any problems since I introduced the bitboards for storing the pieces. Before that, I would have the browser crashing due to high memory usage, and long CPU time would cause the engine to freeze when a move was played. Now, through my testing I’ve confirmed that the performance of the engine is sufficient (with a default depth of 2), and no issues from my testers since the Test 4 with Test User 3.

I was very pleased with the ability of the engine to play down to its opponents, being able to lose to beginner chess players, and being able to beat more experienced chess players like me. The customisation settings were mostly a success, with the blind spots, aggressiveness, and positional play settings were well noticeable and changed how the games were played, however the piece exchanging tendency was less noticeable, but this wasn't in my main objectives. The GUI was also clear and none of my testers had any problems with it while adjusting the engine.

The introduction of the blind spots made the engine play down to opponents very well, however this did lead to a loss of the human-like playstyle of the engine which was picked up by Test User 3.

Objective 3 – The adventure mode

Testing the adventure mode was hard, as it would take near an hour for a beginner chess player to fully complete the adventure mode, however I had no problems with my own end-to-end test and my one game with Test User 2 worked fully at an expected skill level. The display of the opponent and user's name worked great, and the user could change their name with no issues. The level saving also worked correctly, and the finalised round-up of the user's chess adventure shows relevant and interesting information. The file can also be downloaded fully. This was all shown in Test 9. For the review functionality, I had no problems, but Test User 2 gave some comments about more settings for reviewing, including being able to move forward and backwards in the game and possibly play new moves on the board to improve on how they played, however this was out of the scope of the objectives. The shareable functionality also worked well, which was shown in my end-to-end test and Test 8.

Objective 4 – Server API

The relevant API resources are accessible, however there are still some minor problems for showing the user the errors made by the request. Most times when an error is occurred, the user is just redirected to the login page (as most errors are due to authentication errors), however this wouldn't work for a server-side error. I haven't encountered any server-side errors in my testing, but of course there will be some extremely edge cases that I've missed, but these will never occur in normal user activity, only when requests are made directly to the API, which should never happen.

Possible Improvements

Here are the main improvements that I'd make from the issues made in my testing and evaluation.

- One of the missing components from my system is that there is no draw by repetition. This effects the custom game section of the project but doesn't affect a major part of the system, the adventure mode. The adventure mode is not affected, as the player only moves on to the next stage once they win a game, and the engine is coded to evaluate repeated positions much lower, preventing draws in most cases.
- For the actual engine playstyle, the engine doesn't play too human like, which was mentioned by some of my testers, by exclamations of strange moves. Mostly this was caused by poor performance, resulting in lower depth settings, but adding some neural network, trained on real games would have made the engine play a lot more human like.

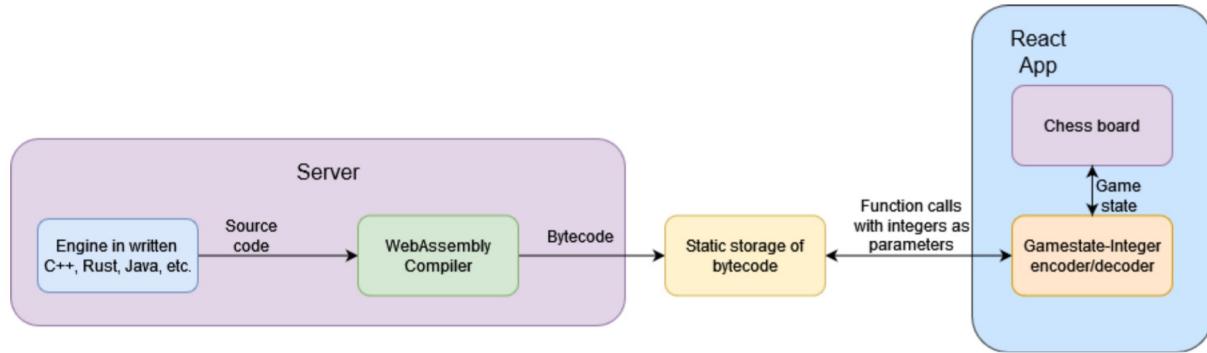
One of the largest bottlenecks of my system is the construction of the JavaScript Bigint, which is used for the square collection class discussed in the design section. To get around this, I would have written the engine in a language which supports strongly typed unsigned 64-bit integers like Java or C and its derivatives, and compile the source code into intermediate WebAssembly bytecode, which could be run directly in the browser.

To add the neural network, I would train the engine on a data set of games which have been analysed by an existing engine. The nodes of the neural network would be the piece positions and the output node would be the evaluation given by the engine. This wouldn't replace the minimax algorithm, but instead would replace the evaluation function which is conducted at each leaf node of the search, with a function which uses the neural network weights to generate an evaluation.

From the advice from my testers, I would also make the website more accessible for navigation. The website still can be navigated by any user, but the constant reloading of the page makes the website feel more chunky, also I would investigate making the website more responsive for touchscreen and different screen resolutions, but that would require a complete redesign of the drag-and-drop behaviour of the board.

A major problem with WebAssembly is that with most translated language, to call a function, the function must only take simple parameters like signed integers. Therefore, we must also be able to encode and decode the board state, as WebAssembly functions must be stateless and not cause side-effects. The encoder could encode the entire board state into integers by concatenating the binary representation of the piece in each square of the board together using binary shifts. We use 5 bits for each square, so 12 squares could be stored in one 64-bit unsigned integer. 6 of these integers could represent the entire board, and other unsigned integers could be used to store the game state, and the past game state stack in similar ways.

Here is a DFD showing what would need to be changed to instead use WebAssembly instead of JavaScript, which would most likely be the most influential change for performance.



For the actual engine code, apart from translating the code from TypeScript to C++, Rust, Java, etc., not much of the design would change. I would certainly make more use of the ability to perform binary operations on integers larger than 32 bits, which would make the Bitboards faster so I would maybe use them more, especially in the evaluation function, as they would carry much less of a performance overhead.

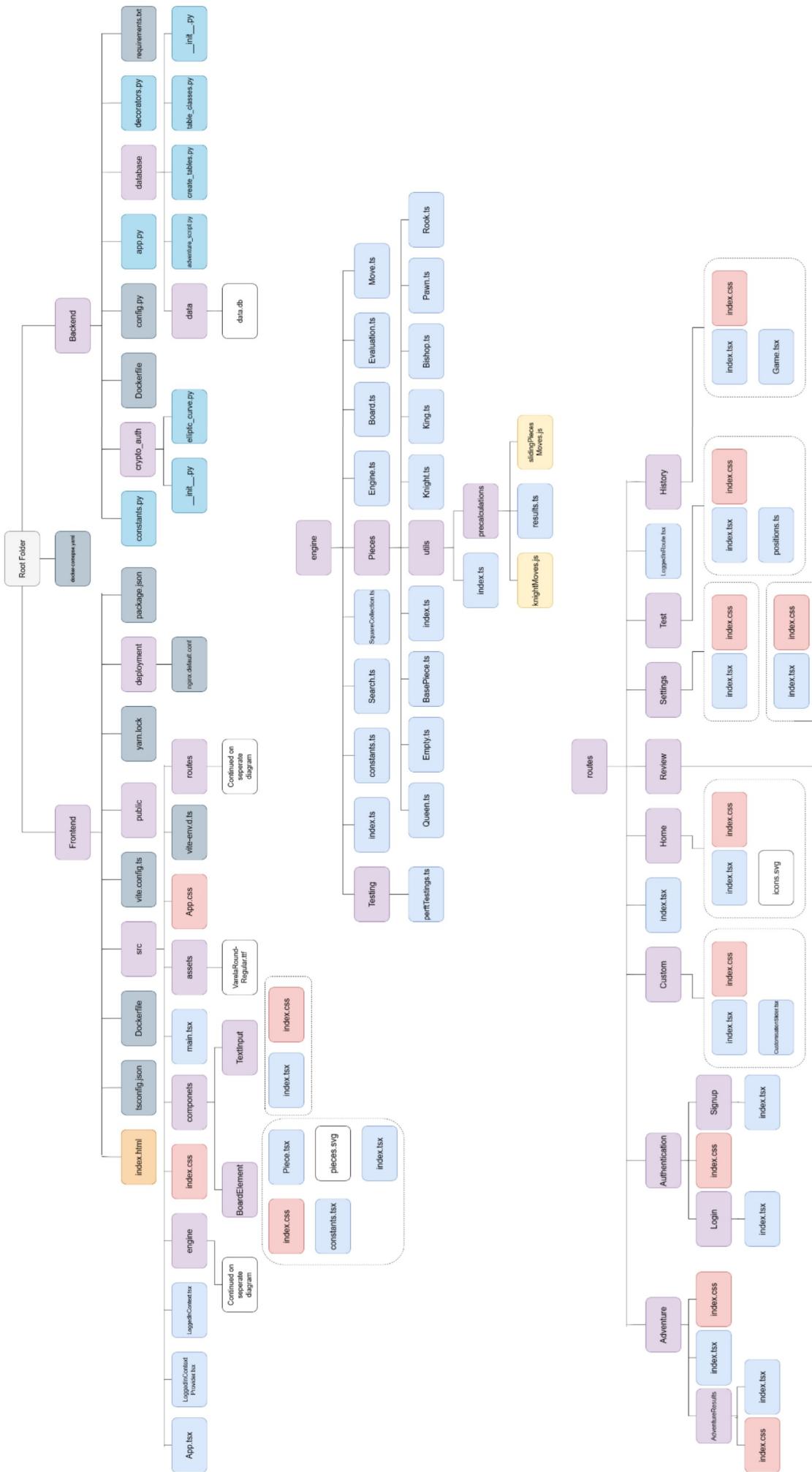
5 Code

File Hierarchy Diagram

On the next page is a hierarchy diagram which shows all the files in the project. The lines between the files and directories represent files within directories not the relationships between files. The files are colour coded (I will give examples for each colour if the chart is being read in black-and-white). The dark grey files are config files which are generally irrelevant to the algorithms and programming structures accessed at A level. An example of this file is the ‘docker-compose.yaml’ file near the top of the chart. Directories are coloured with a purple, for example the ‘Frontend’ file, and TypeScript files are coloured in a light blue, for example index.tsx. TypeScript files ending in .tsx, rather than .ts signifies that the files contains JSX for the GUI structure. Normal JavaScript files are coloured in yellow, which is only the files knightMove.js and slidingPiecesMoves.js. Solid white files represent files which are not meant to be read using a text editor for the purpose of the project (database, SVG files) and I will not be including these files in the write up. Turquoise files such as app.py are files written in Python and red files such as index.css are CSS files used for the styling of the GUI.

The dotted lines surrounding a group of files represent files stored in the same subdirectory which have been grouped to preserve space in the diagram. Parts of the diagram have been split up to stop the diagram from becoming too long, with the frontend/src/routes and frontend/src/engine directories being shown in separate diagrams below. Each file will be given a description, and the contents of the file will be shown.

Files used solely for the development process, like code editor settings, .gitignore, .dockerignore files have been omitted along with font files and SVG files.



File Descriptions

docker-compose.yaml – Config file which configures the Docker container.

frontend – A folder containing the files which are used to create the chess engine and GUI.

frontend/tsconfig.json – Config file which determine how TypeScript is compiled.

frontend/yarn.lock – Config file which keeps package versions constant.

frontend/vite.config.ts – Config files which determines how the Vite project will be built.

frontend/package.json – Config file which says which packages to use.

frontend/Dockerfile – Config file which determines how the frontend image will be built.

frontend/index.html – Almost fully empty HTML file which the React app is loaded into.

frontend/deployment/nginx.default.conf – Sets up how NGINX will handle web requests.

frontend/public – Empty directory which contains the built React app.

frontend/src – A folder containing the actual React app.

frontend/dist – An empty folder that will contain the output from the Vite building process.

frontend/src/main.tsx – The file where the React app is initialised and built from.

frontend/src/App.tsx – The file that imports the routes in the routes folder.

frontend/src/App.css – Stylesheet for styles used in the entire app.

frontend/src/index.css – Stylesheet for styles used in the entire website.

frontend/src/assets/VarelaRound-Regular.ttf – Font used on the website.

frontend/src/vite-env.d.ts – Adds support for TypeScript types for SVGs.

frontend/src/LoggedInContext.tsx – Creates the React context for the user API request.

frontend/src/LoggedInContextProvider.tsx – Makes an API request for the user info on page load.

frontend/src/engine – Folder which contains all the files for the actual chess engine program.

frontend/src/engine/Testing/perftTesting.ts – File which contains an algorithm for running automatic PERFT tests.

frontend/src/engine/index.ts – Dummy file to indicate that the folder should be treated as a package.

frontend/src/engine/constants.ts – File containing constants used in the engine.

frontend/src/engine/Search.ts – File containing the minimax algorithm used to find the best move.

frontend/src/engine/SquareCollection.ts – File containing the SquareCollection class.

frontend/src/engine/Engine.ts – File containing the Engine class.

frontend/src/engine/Board.ts – File containing the Board class.

frontend/src/engine/Move.ts – File containing the Move class.

frontend/src/engine/Evaluation.ts – File containing algorithms to evaluate a position in a leaf node when searching for the best move.

frontend/src/engine/Pieces – Folder containing all piece's classes and move generation algorithms.

frontend/src/engine/Pieces/**index.ts** – File which chooses the Piece class to construct.

frontend/src/engine/Pieces/<Chess Piece>.ts – A group of files where each file contains a class for initialising the piece given in the file name.

frontend/src/engine/Pieces/**Empty.ts** – A class generating a piece which represents an empty square.

frontend/src/engine/Pieces/**BasePiece.ts** – A file containing the abstract base class for each piece.

frontend/src/engine/Pieces/**utils** – Folder for algorithms used for generating moves in the piece classes.

frontend/src/engine/Pieces/utils/**index.ts** – Contains algorithms for generating moves for sliding pieces.

frontend/src/engine/Pieces/utils/**precalculations** – Folder containing files which are used for calculations done before the project is run.

frontend/src/engine/Pieces/utils/precalculations/**knightMoves.js** – File which generates the knight attacks for each square.

frontend/src/engine/Pieces/utils/precalculations/**slidingPiecesMoves.js** – File which generates the sliding piece attacks for each square.

frontend/src/engine/Pieces/utils/precalculations/**results.ts** – File containing the results from the JS files for use in the move generation algorithms.

frontend/src/**components** – Folder containing React classes used across multiple routes.

frontend/src/components/**BoardElement** – Folder containing files for creating the board GUI.

frontend/src/components/BoardElement/**constants.tsx** – File containing constants for use in the rendering of the board given by the engine.

frontend/src/components/BoardElement/**pieces.svg** – File containing SVGs for each piece.

frontend/src/components/BoardElement/**index.css** – File containing styles for creating the board.

frontend/src/components/BoardElement/**Piece.tsx** – React component for each piece.

frontend/src/components/BoardElement/**index.ts** – Component for the board and rendering process.

frontend/src/components/**TextInput** – Folder containing files for the text input component.

frontend/src/components/TextInput/**index.ts** – File containing the text input component.

frontend/src/components/TextInput/**index.css** – File containing the text input component styles.

frontend/src/**routes** – Folder containing components for each route.

frontend/src/routes/**index.tsx** – Folder which imports and reexports the routes for App.tsx.

frontend/src/routes/**LoggedInRoute.tsx** – Folder which handles redirection when not logged in.

frontend/src/routes/(**Adventure/Authentication/Custom/Home/Review/Settings/Test/History**) – Folders which contain the component for each route discussed in the React Component chart earlier.

frontend/src/routes/<route name>/**index.js** – File for each route which contains the constructor for each route page.

frontend/src/routes/<route name>/**index.css** – File which contains the styles for each route.

frontend/src/routes/History/**Game.tsx** – File which contains a component for each element in the history game list.

frontend/src/routes/Test/**positions.ts** – File which contains positions to be PERFT tested.

frontend/src/routes/Home/**icons.svg** – Images in the home route which are just for looking pretty.

frontend/src/routes/Custom/**CustomisationSlider.tsx** – Component for each slider on the Custom route.

frontend/src/routes/Login/**index.tsx** – File for the rendering and checks for the login form.

frontend/src/routes/Signup/**index.tsx** – File for the rendering and checks for the signup form.

frontend/src/routes/Adventure/**AdventureResults** – Folder containing code for generating the component which shows the user statistic sheet when the adventure is completed.

backend – Folder containing the Python Flask app API.

backend/**Dockerfile** – Config file for creating the backend image.

backend/**config.py** – Config file for flask settings, app variables, and keys

backend/**app.py** – Main file for initialising the app and creating the API routes.

backend/**requirements.txt** – File for declaring which packages should be installed.

backend/**decorators.py** – File containing the function decorator for the authentication process.

backend/**database** – Folder containing code related to the database.

backend/database/**__init__.py** – File which contains a database class which acts as an interface for making SQL queries indirectly in app.py on the database.

backend/database/**table_classes.py** – Declares classes used in the __init__ file for passing back to app.py to structure data coming from the database more rigorously.

backend/database/**adventure_script.py** – Exports the story script for inserting into the database.

backend/database/**create_tables.py** – Creates the SQL tables, the relations between them and inserts the adventure script into the campaign levels table.

backend/database/**data/data.db** – SQLite3 Database file

backend/**crypto_auth** – Folder containing algorithms for password hashing and message signing.

backend/crypto_auth/**elliptic_curve.py** – File which defines the elliptic curve class and methods for signing a byte message.

backend/crypto_auth/**__init__.py** – File which contains functions for signing a Python dictionary using the functions on the elliptic curve class and uses the SHA256 algorithm to create password hashes and salts.

Cover Sheet

The raw code for each of these files is in the appendix of this project. The order of the files is (roughly) the order of file description for ease of navigation.

Files and directories of note:

- backend/crypto_auth/**elliptic_curve.py** | Page 158 – Page 161
This file contains all the cryptography related algorithms in a class which I have discussed in pseudocode and flowcharts in the design section extensively.
Complex mathematical operations (group theory and elliptic curves)
- frontend/src/engine/**Board.ts** | Page 82 – Page 89
This file contains the central Board class for the engine and contains many of the algorithms and data structures discussed.
Complex OOP model, aggregation, composition, hashing, stacks, bitfields, dynamic object generation.
- Files in frontend/src/engine/**Pieces** | Page 92 – Page 101
This set of files contains definitions for the OOP model for each chess piece.
Abstract base class, inheritance, polymorphism.
- frontend/src/engine/**Search.ts** | Page 76 – Page 78
This file contains the main algorithm of the project, the alpha-beta minimax algorithm.
Complex-user defined algorithm, insertion sort, recursive algorithms.
- backend/**app.py** | Page 144 – Page 149
Contains functions for the API.
Complex client-server model, server-side scripting using request and response objects, parameterised Web service APIs and parsing JSON.
- frontend/src/engine/**Engine.ts** | Page 79 – Page 82
Contains a class which is the interface between the web app and the engine, also contains algorithms pertaining to the engine customisation.
Interfaces, complex mathematical operations
- backend/**decorators.py** | Page 149 – Page 150
Handles logic for the token scopes and authentication.
Complex client-server model
- backend/**database** | Page 150 – Page 159
Contains the SQL queries talked about earlier and contains a class-based interface between app.py and the database.
Cross-table parameterised SQL, complex data model in database with several interlinked tables

Overall, most of the complexity of the project is contained in the frontend/src/engine folder, and all other files are supporting code for the GUI, the server, and more bells and whistles which differentiates my project from a standard chess engine.

Filetypes

Files ending in .ts are standard TypeScript files.

Files ending in .tsx are TypeScript files which may use or handle JSX.

Files ending in .py are Python files

Files ending in .js are JavaScript files

Files ending in .svg are vector images

Files ending in .css are stylesheets for the GUI

Files ending in .txt, .json, .yaml, .d.ts, .config.ts, .conf, or Dockerfile are config files

Attributions and clarity

As with all programming projects, some of the inspiration for techniques used in the engine were taken from existing sources (Chess Programming Wiki). However, all code below is my own unless commented otherwise. The complex mathematical equations used for the elliptic curve algorithm are not my own to ensure compliance with the FIPS 186-4 standard, however all implementation of the methods are my own. The email regex used is not my own, which is clearly commented in the code as the RFC 822 Compliant email regex by Cal Henderson. Some of the GUI work (the text inputs) and Docker and NGINX configuration were borrowed from a previous project I had done, although it is all still my own code.

Appendix

docker-compose.yaml

```

1. version: '3.2'
2. services:
3.   backend:
4.     build:
5.       context: ./backend
6.       dockerfile: Dockerfile
7.       image: nea-backend
8.       volumes:
9.         - type: volume
10.           source: sqlite-db
11.           target: /backend/database/data
12.   frontend:
13.     build:
14.       context: ./frontend
15.       dockerfile: Dockerfile
16.       image: nea-frontend
17.       ports:
18.         - 8000:80
19.
20. volumes:
21.   sqlite-db:
22.
```

frontend/tsconfig.json

```

1. {
2.   "compilerOptions": {
3.     "target": "ESNext",
4.     "useDefineForClassFields": true,
5.     "lib": [
6.       "DOM",
7.       "DOM.Iterable",
8.       "ESNext"
9.     ],
10.    "allowJs": false,
11.    "skipLibCheck": true,
12.    "esModuleInterop": false,
13.    "allowSyntheticDefaultImports": true,
14.    "strict": true,
15.    "forceConsistentCasingInFileNames": true,
16.    "module": "ESNext",
17.    "moduleResolution": "Node",
18.    "resolveJsonModule": true,
19.    "isolatedModules": true,
20.    "noEmit": true,
21.    "jsx": "react-jsx",
22.  },
23.  "include": [
24.    "src"
25.  ],
26.  "references": [
27.    {
28.      "path": "./tsconfig.node.json"
29.    }
30.  ]
31. }
32.
```

frontend/vite.config.ts

```

1. import { defineConfig } from "vite"
2.
3. export default defineConfig({
4. })
```

frontend/package.json

```

1. {
2.   "name": "frontend",
3.   "private": true,
4.   "version": "0.0.0",
5.   "type": "module",
6.   "scripts": {
7.     "dev": "vite",
8.     "build": "tsc && vite build",
9.     "preview": "vite preview"
10.    },
11.   "dependencies": {
12.     "react": "^18.2.0",
13.     "react-dom": "^18.2.0",
14.     "react-router-dom": "^6.10.0"
15.   },
16.   "devDependencies": {
17.     "@types/react": "^18.0.28",
18.     "@types/react-dom": "^18.0.11",
19.     "@vitejs/plugin-react-swc": "^3.0.0",
20.     "typescript": "^4.9.3",
21.     "vite": "^4.2.0"
22.   }
23. }
24.

```

frontend/Dockerfile

```

1. # node base image
2. FROM node:16-alpine as vite-build
3.
4. # set work directory
5. WORKDIR /frontend
6.
7. # set environment variables
8. ENV PATH ./node_modules/.bin$PATH
9.
10. # copy project
11. COPY . .
12.
13. # install and build vite app
14. RUN yarn install
15. RUN yarn build
16.
17. # start and configure nginx
18. FROM nginx:stable-alpine
19. COPY --from=vite-build /frontend/dist /usr/share/nginx/html
20. COPY /deployment/nginx.default.conf /etc/nginx/conf.d/default.conf
21.

```

frontend/index.html

```

1. <!DOCTYPE html>
2. <html lang="en">
3.
4.   <head>
5.     <meta charset="UTF-8" />
6.     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7.     <title>thochess - Chess Learning</title>
8.   </head>
9.
10.  <body>
11.    <div id="root"></div>
12.    <script type="module" src="/src/main.tsx"></script>

```

```
13. </body>
14.
15. </html>
16.
```

frontend/development/nginx.default.conf

```
1. server {
2.   listen      80;
3.   server_name localhost;
4.
5.   root    /usr/share/nginx/html;
6.   index index.html;
7.   error_page  500 502 503 504  /50x.html;
8.
9.   location / {
10.     try_files $uri /index.html;
11.     add_header Cache-Control "no-cache";
12.   }
13.
14.   location /assets {
15.     expires 1y;
16.     add_header Cache-Control "public";
17.   }
18.
19.   location /api {
20.     proxy_pass http://backend:5000/api;
21.   }
22.
23.   location /s/ {
24.     proxy_pass http://backend:5000/s/;
25.   }
26. }
27.
```

frontend/src/main.tsx

```
1. import React from 'react'
2. import ReactDOM from 'react-dom/client'
3. import { BrowserRouter } from 'react-router-dom'
4. import App from './App'
5.
6. ReactDOM.createRoot(document.getElementById('root') as HTMLElement).render(
7.   <BrowserRouter>
8.     <App />
9.   </BrowserRouter>
10. )
11.
```

frontend/src/App.tsx

```
1. import { Suspense } from 'react'
2. import { Route, Routes } from "react-router-dom";
3.
4. import { routes, loggedInRoutes } from "./routes"
5. import LoggedInRoute from './routes/LoggedInRoute';
6. import LoggedInContextProvider from './LoggedInContextProvider';
7.
8. import './App.css'
9.
10.
11. function App() {
12.   return (
13.     <div className="App">
```

```

14.      <Suspense>
15.        <Routes>
16.          {routes.map((route, index) => (
17.            <Route
18.              key={`path-${route.path}-${index}`}
19.              path={route.path}
20.              element={<route.element />}
21.            />
22.          )))
23.          {loggedInRoutes.map((route, index) => (
24.            <Route key={`path-${route.path}-${index}`} path={route.path}
element={<LoggedInRoute />}>
25.              <Route path={route.path} element={
26.                <LoggedInContextProvider>
27.                  <route.element />
28.                  </LoggedInContextProvider>
29.                } />
30.              </Route>
31.            )))
32.          </Routes>
33.        </Suspense>
34.      </div>
35.    )
36.  }
37.
38. export default App
39.

```

frontend/src/App.css

```

1. @font-face {
2.   font-family: "Varela Round";
3.   font-weight: 400;
4.   font-style: normal;
5.   src: local("Verale Round"), url("./assets/VarelaRound-Regular.ttf") format("truetype");
6. }
7.
8. #root {
9.   margin: 0 auto;
10.  height: 100%;
11.  width: 100%;
12.  font-family: "Varela Round", sans-serif;
13. }
14.
15. html,
16. body,
17. #root,
18. .App {
19.   margin: 0;
20.   height: 100%;
21.   width: 100%;
22. }
23.
24. button {
25.   color: #525151;
26.   background-color: #faf1ff;
27.   border: none;
28.   padding: 10px 20px 10px 20px;
29.   font-size: 1.1em;
30.   border-radius: 20px;
31.   margin-top: 15px;
32.   cursor: pointer;
33. }

```

frontend/src/vite-env.d.ts

```
1. /// <reference types="vite/client" />
```

frontend/src/LoggedInContext.tsx

```
1. import { createContext } from "react"
2.
3. export type LoggedInContextType = { id: number, displayName: string, levelid: string }
4.
5. export const defaultLoggedInContext: LoggedInContextType = {
6.   id: -1,
7.   displayName: "",
8.   levelid: "0"
9. }
10.
11. export const LoggedInContext = createContext(defaultLoggedInContext)
12.
```

frontend/src/LoggedInContextProvider.tsx

```
1. import React from "react"
2. import { LoggedInContext, LoggedInContextType, defaultLoggedInContext } from
"./LoggedInContext"
3.
4. interface Props {
5.   children: React.ReactNode
6. }
7.
8. interface State {
9.   userData: LoggedInContextType
10. }
11.
12. class LoggedInContextProvider extends React.Component<Props, State> {
13.   constructor(props: Props) {
14.     super(props)
15.
16.     this.state = {
17.       userData: defaultLoggedInContext
18.     }
19.   }
20.
21.   componentDidMount() {
22.     fetch("/api/users/@me").then(resp => resp.json()).then(data => {
23.       if (!data.error) {
24.         this.setState({
25.           userData: {
26.             id: data.data.id,
27.             displayName: data.data.name,
28.             levelid: data.data.level_id
29.           }
30.         })
31.       }
32.     })
33.   }
34.
35.   render() {
36.     return (
37.       <LoggedInContext.Provider value={this.state.userData}>
38.         {this.props.children}
39.       </LoggedInContext.Provider>
40.     )
41.   }
42. }
```

```
44. export default LoggedInContextProvider
45.
```

frontend/src/engine/Testing/perftTesting.ts

```
1. import Board from '../Board'
2.
3. // Function implementation copied from a function written in C
4. // https://www.chessprogramming.org/Perft
5. export function perft(depth: number, board: Board) {
6.     const maxDepth = depth
7.
8.     function testAtDepth(depth: number) {
9.         let moveList = []
10.        let nodes = 0
11.
12.        if (depth == 0) {
13.            return 1
14.        }
15.
16.        moveList = board.generateLegalMoves()
17.
18.        for (let i = 0; i < moveList.length; i++) {
19.            board.playMove(moveList[i])
20.            nodes += testAtDepth(depth - 1)
21.            board.unplayMove(moveList[i])
22.        }
23.
24.
25.        return nodes
26.    }
27.    const result = testAtDepth(depth)
28.    return result
29. }
30.
```

frontend/src/engine/index.ts

```
1. import Engine from "./Engine";
2.
3. export default Engine
4.
```

frontend/src/engine/constants.ts

```
1. export enum Pieces {
2.     empty = 0,
3.     pawn,
4.     rook,
5.     knight,
6.     bishop,
7.     queen,
8.     king,
9.     black = 8,
10.    white = 16,
11.    all = 0,
12. }
13.
14. export const StartingBoard = new Uint8Array([
15.     Pieces.white | Pieces.rook,
16.     Pieces.white | Pieces.knight,
17.     Pieces.white | Pieces.bishop,
18.     Pieces.white | Pieces.queen,
19.     Pieces.white | Pieces.queen,
```

```

20.    Pieces.white | Pieces.bishop,
21.    Pieces.white | Pieces.knight,
22.    Pieces.white | Pieces.rook,
23.    ... (new Array(8).fill(Pieces.white | Pieces.pawn)),
24.    ... (new Array(32).fill(Pieces.empty)),
25.    ... (new Array(8).fill(Pieces.black | Pieces.pawn)),
26.    Pieces.black | Pieces.rook,
27.    Pieces.black | Pieces.knight,
28.    Pieces.black | Pieces.bishop,
29.    Pieces.black | Pieces.queen,
30.    Pieces.black | Pieces.king,
31.    Pieces.black | Pieces.bishop,
32.    Pieces.black | Pieces.knight,
33.    Pieces.black | Pieces.rook,
34.  ])
35.
36.
37. // From https://www.chessprogramming.org/Simplified_Evaluation_Function
38. export const PieceSquareTables = {
39.   [Pieces.pawn]: [
40.     0, 0, 0, 0, 0, 0, 0, 0,
41.     50, 50, 50, 50, 50, 50, 50, 50,
42.     10, 10, 20, 30, 30, 20, 10, 10,
43.     5, 5, 10, 25, 25, 10, 5, 5,
44.     0, 0, 0, 20, 20, 0, 0, 0,
45.     5, -5, -10, 0, 0, -10, -5, 5,
46.     5, 10, 10, -40, -40, 10, 10, 5,
47.     0, 0, 0, 0, 0, 0, 0, 0
48.   ],
49.   [Pieces.knight]: [
50.     -50, -40, -30, -30, -30, -40, -50,
51.     -40, -20, 0, 0, 0, 0, -20, -40,
52.     -30, 0, 10, 15, 15, 10, 0, -30,
53.     -30, 5, 15, 20, 20, 15, 5, -30,
54.     -30, 0, 15, 20, 20, 15, 0, -30,
55.     -30, 5, 10, 15, 15, 10, 5, -30,
56.     -40, -20, 0, 5, 5, 0, -20, -40,
57.     -50, -40, -30, -30, -30, -40, -50
58.   ],
59.   [Pieces.bishop]: [
60.     -20, -10, -10, -10, -10, -10, -20,
61.     -10, 0, 0, 0, 0, 0, -10,
62.     -10, 0, 5, 10, 10, 5, 0, -10,
63.     -10, 5, 5, 10, 10, 5, 5, -10,
64.     -10, 0, 10, 10, 10, 10, 0, -10,
65.     -10, 10, 10, 10, 10, 10, 10, -10,
66.     -10, 5, 0, 0, 0, 0, 5, -10,
67.     -20, -10, -10, -10, -10, -10, -20
68.   ],
69.   [Pieces.rook]: [
70.     0, 0, 0, 0, 0, 0, 0, 0,
71.     5, 10, 10, 10, 10, 10, 10, 5,
72.     -5, 0, 0, 0, 0, 0, 0, -5,
73.     -5, 0, 0, 0, 0, 0, 0, -5,
74.     -5, 0, 0, 0, 0, 0, 0, -5,
75.     -5, 0, 0, 0, 0, 0, 0, -5,
76.     -5, 0, 0, 0, 0, 0, 0, -5,
77.     0, 0, 0, 5, 5, 0, 0, 0
78.   ],
79.   [Pieces.queen]: [
80.     -20, -10, -10, -5, -5, -10, -10, -20,
81.     -10, 0, 0, 0, 0, 0, 0, -10,
82.     -10, 0, 5, 5, 5, 5, 0, -10,
83.     -5, 0, 5, 5, 5, 5, 0, -5,
84.     0, 0, 5, 5, 5, 5, 0, -5,
85.     -10, 5, 5, 5, 5, 5, 0, -10,
86.     -10, 0, 5, 0, 0, 0, 0, -10,
87.     -20, -10, -10, -5, -5, -10, -10, -20
88.   ],
89.   [Pieces.king]: [

```

```

90.      -30, -40, -40, -50, -50, -40, -40, -30,
91.      -30, -40, -40, -50, -50, -40, -40, -30,
92.      -30, -40, -40, -50, -50, -40, -40, -30,
93.      -30, -40, -40, -50, -50, -40, -40, -30,
94.      -20, -30, -30, -40, -40, -30, -30, -20,
95.      -10, -20, -20, -20, -20, -20, -10,
96.      20, 20, 0, 0, 0, 20, 20,
97.      20, 30, 10, 0, 0, 10, 30, 20
98. ],
99. [Pieces.king + 1]: [
100.     -50, -40, -30, -20, -20, -30, -40, -50,
101.     -30, -20, -10, 0, 0, -10, -20, -30,
102.     -30, -10, 20, 30, 30, 20, -10, -30,
103.     -30, -10, 30, 40, 40, 30, -10, -30,
104.     -30, -10, 30, 40, 40, 30, -10, -30,
105.     -30, -10, 20, 30, 30, 20, -10, -30,
106.     -30, -30, 0, 0, 0, 0, -30, -30,
107.     -50, -30, -30, -30, -30, -30, -30, -50
108. ]
109. }
110.

```

frontend/src/engine/Search.ts

```

1. import Board from "./Board";
2. import Evaluation from "./Evaluation";
3. import Move from "./Move";
4. import { pieceValue } from "./Evaluation";
5. import { Customisation, defaultCustomisation } from "./Engine";
6. import { PieceSquareTable } from "./Engine";
7.
8. function sortMoves(board: Board, moves: Array<Move>) {
9.     // Length of move array is usually around 30 per position so insertion sort is the
choice for optimisation
10.    // We sort the moveGoodnessArray and the newOrder at the same time so we can just
return the indexes of the moves which should be looked at first
11.    const moveGoodnessArray = moves.map(move => getEstimatedMoveGoodness(board, move))
12.    const newOrder = Array(moves.length)
13.
14.    for (let i = 0; i < newOrder.length; i++) {
15.        newOrder[i] = i
16.    }
17.
18.    let i = 0
19.
20.    while (i < moves.length) {
21.        let j = i
22.        while (j > 0 && moveGoodnessArray[j - 1] > moveGoodnessArray[j]) {
23.            [newOrder[j], newOrder[j - 1]] = [newOrder[j - 1], newOrder[j]];
24.            [moveGoodnessArray[j], moveGoodnessArray[j - 1]] = [moveGoodnessArray[j - 1],
moveGoodnessArray[j]];
25.
26.            j -= 1
27.        }
28.        i += 1
29.    }
30.
31.    return newOrder.reverse()
32. }
33.
34. function getEstimatedMoveGoodness(board: Board, move: Move) {
35.     let estimatedMoveGoodness = 0
36.
37.     const destinationPiece = board.getSquares()[move.getDestinationSquare()]
38.     const sourcePiece = board.getSquares()[move.getSourceSquare()]
39.
40.     // Capture difference

```

```
41.     estimatedMoveGoodness += Math.max(pieceValue[destinationPiece.getType()] -
pieceValue[sourcePiece.getType()], 0)
42.
43.     if (move.getFlag() & 0b1000) {
44.         estimatedMoveGoodness += pieceValue[move.getPromotionPiece()]
45.     }
46.     if (move.isCapture()) {
47.         estimatedMoveGoodness += 500
48.     }
49.
50.     return estimatedMoveGoodness
51. }
52.
53. function simplifyPosition(board: Board, alpha: number, beta: number, customisation:
Customisation, pieceSquareTables: PieceSquareTable) {
54.     const evaluation = Evaluation(board, customisation, pieceSquareTables)
55.
56.     if (evaluation >= beta) {
57.         return beta
58.     }
59.     if (evaluation > alpha) {
60.         alpha = evaluation
61.     }
62.
63.     const captures = board.generateLegalMoves().filter(move => move.isCapture())
64.
65.     const estimatedMoveOrder = sortMoves(board, captures)
66.
67.     for (let i = 0; i < captures.length; i++) {
68.         board.playMove(captures[estimatedMoveOrder[i]])
69.         const evaluation = -simplifyPosition(board, -beta, -alpha, customisation,
pieceSquareTables)
70.         board.unplayMove(captures[estimatedMoveOrder[i]])
71.
72.         if (evaluation >= beta) {
73.             return beta
74.         }
75.         if (evaluation > alpha) {
76.             alpha = evaluation
77.         }
78.     }
79.
80.     return alpha
81. }
82.
83. function search(board: Board, customisation: Customisation = defaultCustomisation,
pieceSquareTables: PieceSquareTable) {
84.     let bestMove = new Move(0) // Dummy move
85.     const maxDepth = customisation.depth
86.     const checkmateEval = -9999999999
87.
88.     function searchDepth(board: Board, depth: number, alpha: number, beta: number) {
89.         if (depth == 0) {
90.             return simplifyPosition(board, alpha, beta, customisation, pieceSquareTables)
91.         }
92.
93.         const moves = board.generateLegalMoves()
94.
95.         if (moves.length === 0) {
96.             if (board.isCheck()) {
97.                 return checkmateEval + depth // Checkmate
98.             }
99.             return 0 // Stalemate
100.        }
101.
102.        let seenMoves = moves.filter(_ => {
103.            return Math.random() * 100 > customisation.blindSpots * (1 + ((depth - 1) /
maxDepth))
104.        })
105.
```

```

106.     let estimatedMoveOrder: Array<number>
107.
108.     if (seenMoves.length === 0) {
109.         estimatedMoveOrder = [0]
110.         seenMoves = [moves[0]]
111.     }
112.     else {
113.         estimatedMoveOrder = sortMoves(board, seenMoves)
114.     }
115.
116.
117.     for (let i = 0; i < estimatedMoveOrder.length; i++) {
118.         const move = seenMoves[estimatedMoveOrder[i]]
119.         board.playMove(move)
120.
121.         let evaluation;
122.
123.         if (board.hasPositionOccurredBefore() && depth < maxDepth) {
124.             board.unplayMove(move)
125.             return 0 // Cut the branch as a draw
126.         }
127.         else {
128.             evaluation = -searchDepth(board, depth - 1, -beta, -alpha)
129.         }
130.
131.         board.unplayMove(move)
132.
133.         if (evaluation >= beta) {
134.             if (depth === maxDepth) {
135.                 bestMove = move
136.             }
137.             // Cut this branch. This branch is now a leaf. The move before was too
138.             // good, so our opponent will never get to this position.
139.             return beta
140.         }
141.
142.         if (evaluation > alpha) {
143.             if (depth === maxDepth) {
144.                 bestMove = move
145.             }
146.             alpha = evaluation
147.         }
148.
149.         return alpha
150.     }
151.
152.     searchDepth(board, maxDepth, -Infinity, Infinity)
153.
154.     if (bestMove.datum === 0) { // Check if it is still the dummy move
155.         throw new Error("Move calculated was invalid!")
156.     }
157.
158.     return bestMove
159. }
160.
161. export default search
162.

```

frontend/src/engine/SquareCollection.ts

```

1. export default class SquareCollection {
2.     private bitboard: bigint
3.     private iteratingBoard: bigint
4.
5.
6.     constructor(bitboard: bigint = 0n) {
7.         this.bitboard = bitboard

```

```

8.         this.iteratingBoard = bitboard
9.     }
10.
11.    getBitboard() {
12.        return this.bitboard
13.    }
14.
15.    add(square: number) {
16.        this.bitboard |= 1n << BigInt(square)
17.    }
18.
19.    remove(square: number) {
20.        this.bitboard &= ~(1n << BigInt(square))
21.    }
22.
23.    or(collection: SquareCollection) {
24.        return new SquareCollection(this.bitboard | collection.bitboard)
25.    }
26.
27.    and(collection: SquareCollection) {
28.        return new SquareCollection(this.bitboard & collection.bitboard)
29.    }
30.
31.    not() {
32.        // Not the first 64 bits of the bitboard (we can't use ~ because bigint is a signed
33.        // 2's complement number)
34.        return new SquareCollection(~this.bitboard & 0xffffffffffffffffn)
35.    }
36.
37.    *[Symbol.iterator]() {
38.        for (let i = 0; i < 64; i++) {
39.            if (i == 0) {
40.                this.iteratingBoard = this.bitboard
41.            }
42.            if (this.iteratingBoard & 1n) {
43.                yield i
44.            }
45.            this.iteratingBoard >>= 1n
46.        }
47.        this.iteratingBoard = this.bitboard
48.    }
49. }
50.

```

frontend/src/engine/Engine.ts

```

1. import Board from './Board'
2. import Move, { Pieces } from './Move'
3. import Search from './Search'
4. import { StartingBoard } from './constants'
5. import { PieceSquareTables } from './constants'
6.
7. export interface Customisation {
8.     depth: number
9.     aggressiveness: number,
10.    tradeHappy: number
11.    positionalPlay: number
12.    blindSpots: number
13. }
14.
15. export const defaultCustomisation: Customisation = {
16.     depth: 3,
17.     positionalPlay: 100,
18.     aggressiveness: 50,
19.     tradeHappy: 50,
20.     blindSpots: 0
21. }

```

```
22.
23. export type PieceSquareTable = Array<{ [key: number]: Array<number> }>
24.
25. class Engine {
26.   public board: Board
27.   private customisation: Customisation
28.   private pieceSquareTables: PieceSquareTable
29.   private moveHistory: Array<Move>
30.
31.
32.   static fromStartingPosition(customisation: Customisation = defaultCustomisation) {
33.     const board = new Board(StartingBoard, 0x000)
34.     return new this(board, customisation)
35.   }
36.
37.   constructor(board: Board, customisation: Customisation = defaultCustomisation,
moveHistory = []) {
38.     this.board = board
39.     this.customisation = customisation
40.     this.moveHistory = moveHistory
41.
42.     let purePieceSquareTables: PieceSquareTable = [{}, PieceSquareTables]
43.
44.     for (let i = 0; i < Object.keys(PieceSquareTables).length; i++) {
45.       const table = PieceSquareTables[i + 1]
46.
47.       let reversedTable = []
48.
49.       for (let j = 0; j < table.length; j++) {
50.         reversedTable.push(table[table.length - 1 - j])
51.       }
52.
53.       purePieceSquareTables[0][i + 1] = reversedTable
54.     }
55.
56.     this.pieceSquareTables = this.addNormalNoise(purePieceSquareTables)
57.   }
58.
59.   public getCustomisation() {
60.     return this.customisation
61.   }
62.
63.   public setAggression(sideToBeAggressive: number): void {
64.     // Add aggression for own table only
65.     const tableIndex: number = sideToBeAggressive === Pieces.white ? 0 : 1
66.
67.     for (const key in this.pieceSquareTables[tableIndex]) {
68.       this.pieceSquareTables[tableIndex][key] =
this.pieceSquareTables[tableIndex][key]
69.         .map((value, index) => {
70.           return value + Math.floor(sideToBeAggressive ? (index / 8) : (63 -
index / 8)) * ((this.customisation.aggressiveness - 50) / 100) * 20
71.         })
72.       }
73.     }
74.
75.     private addNormalNoise(tables: PieceSquareTable): PieceSquareTable {
76.       // We will add normally distributed noise to each table depending on the
positionalPlay value
77.
78.       // Get a random number in the range (-∞,∞) distributed N(0,1)
79.       const StandardNormalSample = () => {
80.         const u1 = Math.random()
81.         const u2 = Math.random()
82.
83.         // Use the Box-Muller Transform, which is bijection from [0,1]^2 to (-∞,∞) with
a standard normal distribution
84.         // https://en.wikipedia.org/wiki/Box%25E2%80%93Muller_transform
85.         return Math.sqrt(-2 * Math.log(u1)) * Math.cos(u2 * 2 * Math.PI)
86.       }
87.
```

```

87.
88.         // If this.customisation.positionalPlay is 100 then variance = 0 so no noise should
be added
89.         if (this.customisation.positionalPlay === 100) {
90.             return tables
91.         }
92.
93.         // When strength == 0, an addition of 50 noise should only happen in 1% of cases
94.         // So if X ~ N(0, variance_max) then P(X > 50) = 0.01
95.         // (X - u)/sd = Z
96.         // 50/sd_max = 2.3263 (from a level maths formula booklet)
97.         // sd_max = 50/2.3263 = 21.49
98.         // We'll say sd_max should be about 20 for the weakest engine
99.         // f: [0, 100] -> (0, 20]
100.        // We'll choose a inverse linear relationship between strength and standard
deviation, so f(x) = 20 - x/5
101.        const standardDeviation = 20 - this.customisation.positionalPlay / 5
102.
103.        tables.forEach(table => {
104.            for (const key in table) {
105.                table[key] = table[key].map(value => {
106.                    return value + (StandardNormalSample() * standardDeviation)
107.                })
108.            }
109.        })
110.
111.        return tables
112.    }
113.
114.
115.    getBestMove(): Move {
116.        return Search(this.board, this.customisation, this.pieceSquareTables)
117.    }
118.
119.    async computerMove(): Promise<Move> {
120.        return new Promise((resolve) => {
121.            setTimeout(() => {
122.                const move = this.getBestMove()
123.                this.board.playMove(move)
124.                this.moveHistory.push(move)
125.                resolve(move)
126.            }, 20)
127.        })
128.    }
129.
130.    playerUCIMove(from: number, to: number): Move {
131.        const move = this.board.playUCIMove(from, to)
132.        this.moveHistory.push(move)
133.
134.        return move
135.    }
136.
137.    getMoveListString(): string {
138.        return this.moveHistory.reduce((moveString, currMove) => {
139.            return moveString +
` ${decToCoord(currMove.getSourceSquare())}${decToCoord(currMove.getDestinationSquare())} `
140.            }, "").slice(0, -1) // Remove extra space at the end
141.    }
142.
143.    static moveHistoryStringToUCI(string: string): number[][] {
144.        return string.split(" ").map(move => {
145.            const sourceSquare = move.slice(0, 2)
146.            const destSquare = move.slice(-2)
147.
148.            return [coordTodec(sourceSquare), coordTodec(destSquare)]
149.        })
150.    }
151. }
152.
153. function coordTodec(coord: string): number {

```

```

154.     const file = coord.charCodeAt(0) - 97
155.     const rank = Number(coord[1]) - 1
156.
157.     return rank * 8 + file
158. }
159.
160. function decToCoord(square: number): string {
161.     return `${String.fromCharCode(97 + (square % 8))}${Math.floor(square / 8) + 1}`
162. }
163.
164. export default Engine
165.

```

frontend/src/engine/Board.ts

```

1. import { Pieces } from './constants'
2. import Move from './Move'
3. import SquareCollection from './SquareCollection'
4. import Piece from './Pieces'
5. import BasePiece from './Pieces/BasePiece'
6.
7. class Board {
8.     private sideToMove: number
9.     private sideToMoveIndex: number
10.    private pastBoards: Array<number>
11.    private epFile: number
12.    private castlingRights: number
13.    private pieceCapturedPlyBefore: number
14.    private pastGameStateStack: Array<number>
15.
16.    private square: Array<BasePiece>
17.    private collections: Array<Array<SquareCollection>>
18.
19.    constructor(binaryBoard: Uint8Array, gameState: number) {
20.        this.pastGameStateStack = [] // LIFO stack
21.        this.pastBoards = [] // LIFO stack
22.
23.        this.sideToMoveIndex = (gameState & 0b1) // 0 is white and 1 is black
24.        this.sideToMove = this.sideToMoveIndex ? Pieces.black : Pieces.white
25.        this.epFile = (gameState & 0b11110) >> 1
26.        this.castlingRights = (gameState & 0b111100000) >> 5
27.        this.pieceCapturedPlyBefore = (gameState & 0b111000000000) >> 9
28.
29.        this.square = Array(64).fill(new Piece.Empty())
30.
31.        this.collections = [
32.            [new SquareCollection(), new SquareCollection()], // All      : index 0
33.            [new SquareCollection(), new SquareCollection()], // Pawns    : index 1
34.            [new SquareCollection(), new SquareCollection()], // Rooks    : index 2
35.            [new SquareCollection(), new SquareCollection()], // Knights   : index 3
36.            [new SquareCollection(), new SquareCollection()], // Bishops   : index 4
37.            [new SquareCollection(), new SquareCollection()], // Queens   : index 5
38.            [new SquareCollection(), new SquareCollection()], // Kings    : index 6
39.        ]
40.
41.
42.        for (let i = 0; i < binaryBoard.length; i++) {
43.            this.square[i] = Piece.FromBinary(binaryBoard[i])
44.
45.            if (binaryBoard[i] !== 0) {
46.                const colourIndex = +this.square[i].isColour(Pieces.black)
47.                this.collections[this.square[i].getType()][colourIndex].add(i)
48.            }
49.        }
50.
51.        this.updateAllPieceCollection()
52.    }
53.

```

```

54.     hasPositionOccurredBefore(): boolean {
55.         return this.pastBoards.includes(this.hashBoard())
56.     }
57.
58.     getCollections(): Array<Array<SquareCollection>> {
59.         return this.collections
60.     }
61.
62.     getEpFile(): number {
63.         return this.epFile
64.     }
65.
66.     getSquares(): Array<BasePiece> {
67.         return this.square
68.     }
69.
70.     getSideToMove(): number {
71.         return this.sideToMove
72.     }
73.
74.     getSideToMoveIndex(): number {
75.         return this.sideToMoveIndex
76.     }
77.
78.     getCastlingRights(): number {
79.         return this.castlingRights
80.     }
81.
82.     private updateAllPieceCollection() {
83.         this.collections[Pieces.all] = [new SquareCollection(), new SquareCollection()]
84.         for (let i = 1; i < this.collections.length; i++) {
85.             for (let j = 0; j < 2; j++) {
86.                 this.collections[Pieces.all][j] =
this.collections[Pieces.all][j].or(this.collections[i][j])
87.             }
88.         }
89.     }
90.
91.     getGameState() {
92.         return this.sideToMoveIndex | (this.epFile << 1) | (this.castlingRights << 5) |
(this.pieceCapturedPlyBefore << 9)
93.     }
94.
95.     playUCIMove(from: number, to: number) {
96.         let move: Move
97.
98.         const movedPiece = this.square[from]
99.         const destPiece = this.square[to]
100.
101.        // Double pawn move - If we're moving a pawn and it moves 16 squares
102.        if (movedPiece.getType() == Pieces.pawn && Math.abs(from - to) === 16) {
103.            move = Move.fromCharacteristics(to, from, false, true)
104.        }
105.
106.        // En passant - If we're moving a pawn and it moves diagonally to an empty square
107.        else if (movedPiece.getType() == Pieces.pawn && destPiece.getType() == Pieces.empty
&& [7, 9].includes(Math.abs(from - to))) {
108.            move = Move.fromCharacteristics(to, from, true, false, true)
109.        }
110.
111.        // Promotion
112.        else if (movedPiece.getType() == Pieces.pawn && Math.floor(to / 8) % 7 == 0) {
113.            if (this.square[to].getType() === Pieces.empty) {
114.                move = Move.fromCharacteristics(to, from, false, false, 0, 0b011) // Always promote to a queen (for the user)
115.            }
116.            else {
117.                move = Move.fromCharacteristics(to, from, true, false, false, 0, 0b011)
118.            }
119.        }

```

```

120.
121.        // Castling KS
122.        else if (movedPiece.getType() == Pieces.King && (to - from) === 2) {
123.            move = Move.fromCharacteristics(to, from, false, false, false, 1)
124.        }
125.        // Castling QS
126.        else if (movedPiece.getType() == Pieces.King && (to - from) === -2) {
127.            move = Move.fromCharacteristics(to, from, false, false, false, 2)
128.        }
129.
130.        // Capture
131.        else if (this.square[to].getType() !== Pieces.empty) {
132.            move = Move.fromCharacteristics(to, from, true)
133.        }
134.        // Quiet Move
135.        else {
136.            move = Move.fromCharacteristics(to, from)
137.        }
138.
139.        this.playMove(move)
140.
141.        return move
142.    }
143.
144.    private hashBoard(): number {
145.        return this.square.reduce((running_total, currentPiece, index) => {
146.            return running_total ^ (currentPiece.datum << (index >> 1)) // max index is 63,
so max (index >> 1) is 31, within the javascript limit of 32 bits for bitwise operations
147.        }, this.getGameState())
148.    }
149.
150.    playMove(move: Move) {
151.        // Calculate the current game state and push it to the top of the pastGameState
stack
152.        this.pastGameStateStack.push(this.getGameState())
153.        this.pastBoards.push(this.hashBoard())
154.
155.        const flag = move.getFlag()
156.        const dest = move.getDestinationSquare()
157.        const source = move.getSourceSquare()
158.
159.        const piece = this.square[source]
160.
161.        const colourOffset = piece.isColour(Pieces.white) ? 0 : 56
162.
163.        this.pieceCapturedPlyBefore = Pieces.empty
164.
165.        // Remove piece from source square
// In the square-oriented data structure
166.        this.square[source] = new Piece.Empty()
167.
168.        // And in the piece-oriented data structure
169.        this.collections[piece.getType()][piece.getColourIndex()].remove(source)
170.
171.        // Add the piece to the destination square. We will have to check for flag
scenarios, however.
172.
173.        // En passant flag
174.        if (flag === 0b0101) {
175.            const enPassantSquare = dest + (piece.isColour(Pieces.white) ? -8 : 8)
176.
177.            // Only pawns can be en-passanted
178.
179.            this.collections[Pieces.pawn][piece.getOpponentColourIndex()].remove(enPassantSquare)
180.
181.            this.square[enPassantSquare] = new Piece.Empty()
182.        }
183.
184.        // Castling - KS
185.        else if (flag === 0b0010) {

```

```

186.         this.collections[Pieces.rook][piece.getColourIndex()].remove(7 + colourOffset)
187.         this.collections[Pieces.rook][piece.getColourIndex()].add(5 + colourOffset)
188.
189.         this.square[7 + colourOffset] = new Piece.Empty()
190.         this.square[5 + colourOffset] = new Piece.Rook(piece.getColour())
191.     }
192.
193.     // Castling - QS
194.     else if (flag === 0b0011) {
195.         this.collections[Pieces.rook][piece.getColourIndex()].remove(0 + colourOffset)
196.         this.collections[Pieces.rook][piece.getColourIndex()].add(3 + colourOffset)
197.
198.         this.square[0 + colourOffset] = new Piece.Empty()
199.         this.square[3 + colourOffset] = new Piece.Rook(piece.getColour())
200.     }
201.
202.
203.     // If the move is a capture we will have to remove the captured piece from its
204.     // collection (unless en-passant, but that is handled separately)
205.     else if (move.isCapture()) {
206.         const capturedPiece = this.square[dest]
207.
208.         if (capturedPiece.getType() === Pieces.rook) {
209.             const shift = capturedPiece.isColour(Pieces.white) ? 0 : 2
210.
211.             if (dest + colourOffset - 56 === 7) {
212.                 // Remove KS castling rights
213.                 this.castlingRights |= 0b0001 << shift
214.             }
215.             else if (dest + colourOffset - 56 === 0) {
216.                 // Remove QS castling rights
217.                 this.castlingRights |= 0b0010 << shift
218.             }
219.         }
220.
221.         this.collections[capturedPiece.getType()][capturedPiece.getColourIndex()].remove(dest)
222.         this.pieceCapturedPlyBefore = capturedPiece.getType()
223.     }
224.
225.     // Check if the rook or king has moved (we will remove castling rights) This will
226.     // also take care of removing castling rights when castling
227.     if (piece.getType() === Pieces.king) {
228.         // Remove all castling rights
229.         const shift = piece.isColour(Pieces.white) ? 0 : 2
230.         this.castlingRights |= 0b0011 << shift
231.     }
232.     else if (piece.getType() === Pieces.rook) {
233.         const shift = piece.isColour(Pieces.white) ? 0 : 2
234.
235.         if (source - colourOffset === 7) {
236.             // Remove KS castling rights
237.             this.castlingRights |= 0b0001 << shift
238.         }
239.         if (source - colourOffset === 0) {
240.             // Remove QS castling rights
241.             this.castlingRights |= 0b0010 << shift
242.         }
243.
244.         // Check if we're promoting
245.         let destinationPiece = piece
246.
247.         if (move.isPromotion()) {
248.             destinationPiece = Piece.FromBinary(piece.getColour() |
249.             move.getPromotionPiece())
250.         }
251.
252.         // Add the piece in the square-oriented data structure. This will automatically
253.         // take care of non-enpassant captures

```

```

251.     this.square[dest] = destinationPiece
252.     // Add the piece into the piece-oriented data structure
253.
254.     this.collections[destinationPiece.getType()][destinationPiece.getColourIndex()].add(dest)
255.
256.     // Double pawn push means en passant is possible for the next turn on that file
257.     if (flag === 0b0001) {
258.         this.epFile = (dest % 8) + 1
259.     }
260.     else {
261.         this.epFile = 0
262.     }
263.
264.     this.updateAllPieceCollection()
265.
266.     this.sideToMoveIndex = 1 - this.sideToMoveIndex
267.     this.sideToMove = this.sideToMoveIndex ? Pieces.black : Pieces.white
268.
269.     unplayMove(move) {
270.         // Remove the most recent gamestate
271.         const newGameState = this.pastGameStateStack.pop()
272.         this.pastBoards.pop()
273.
274.         if (newGameState === undefined) {
275.             throw "No move to unplay!"
276.         }
277.
278.         this.sideToMoveIndex = (newGameState & 0b1)
279.         this.sideToMove = this.sideToMoveIndex ? Pieces.black : Pieces.white
280.
281.         const opponentMoveIndex = 1 - this.sideToMoveIndex
282.         const opponentColour = this.sideToMoveIndex ? Pieces.white : Pieces.black
283.
284.         const flag = move.getFlag()
285.         const dest = move.getDestinationSquare()
286.         const source = move.getSourceSquare()
287.
288.
289.         const pieceCapturedLastPly = this.pieceCapturedPlyBefore === Pieces.empty ?
290.             Pieces.empty : this.pieceCapturedPlyBefore | opponentColour
291.
292.         let piece = this.square[dest]
293.
294.         const colourOffset = piece.isColour(Pieces.white) ? 0 : 56
295.
296.         // Removed moved piece from data structures and replace it with an empty square or
297.         // the captured piece
298.         this.square[dest] = Piece.FromBinary(pieceCapturedLastPly)
299.         this.collections[piece.getType()][this.sideToMoveIndex].remove(dest)
300.
301.         // Handle en passant
302.         if (move.isEnPassant()) {
303.             const enPassantOffset = piece.isColour(Pieces.white) ? -8 : 8
304.
305.             this.square[dest + enPassantOffset] = new Piece.Pawn(opponentColour)
306.             this.collections[Pieces.pawn][opponentMoveIndex].add(dest + enPassantOffset)
307.
308.         } else if (move.isPromotion()) {
309.             piece = new Piece.Pawn(this.sideToMove)
310.
311.         } // KS castling
312.         else if (flag === 0b0010) {
313.             this.collections[Pieces.rook][this.sideToMoveIndex].remove(5 + colourOffset)
314.             this.collections[Pieces.rook][this.sideToMoveIndex].add(7 + colourOffset)
315.
316.             this.square[5 + colourOffset] = new Piece.Empty()
317.             this.square[7 + colourOffset] = new Piece.Rook(this.sideToMove)
318.
319.         } // QS castling

```

```

318.         else if (flag === 0b0011) {
319.             this.collections[Pieces.rook][this.sideToMoveIndex].remove(3 + colourOffset)
320.             this.collections[Pieces.rook][this.sideToMoveIndex].add(0 + colourOffset)
321.
322.             this.square[3 + colourOffset] = new Piece.Empty()
323.             this.square[0 + colourOffset] = new Piece.Rook(this.sideToMove)
324.         }
325.         // Will also take care of promotion captures
326.         if (move.isCapture() && !move.isEnPassant()) {
327.             this.collections[pieceCapturedLastPly & 0b111][opponentMoveIndex].add(dest)
328.         }
329.
330.         // Source square replace with the moved piece
331.         this.square[source] = piece
332.         this.collections[piece.getType()][this.sideToMoveIndex].add(source)
333.
334.         this.castlingRights = (newGameState & 0b111100000) >> 5
335.         this.pieceCapturedPlyBefore = (newGameState & 0b1110000000000) >> 9
336.         this.epFile = (newGameState & 0b11110) >> 1
337.         this.updateAllPieceCollection()
338.     }
339.
340.     isSquareAttacked(square: number, attackerColour: number): boolean {
341.         const attackerColourIndex = attackerColour === Pieces.white ? 0 : 1
342.         const defenderColour = attackerColourIndex == 0 ? Pieces.black : Pieces.white
343.         const blockers = this.collections[Pieces.all][attackerColourIndex].getBitboard() |
344.             this.collections[Pieces.all][1 - attackerColourIndex].getBitboard()
345.
346.         // Looping over all pieces binary values
347.         for (let i = 1; i < Pieces.king + 1; i++) {
348.             const piece = Piece.FromBinary(i | defenderColour)
349.
350.             if (piece.getAttacks(square, blockers) &
351.                 this.collections[i][attackerColourIndex].getBitboard()) {
352.                 return true
353.             }
354.
355.             return false
356.         }
357.
358.         generateLegalMoves(): Array<Move> {
359.             let moveList: Array<Move> = []
360.
361.             for (let i = 0; i < 64; i++) {
362.                 if (this.square[i].isColour(this.sideToMove)) {
363.                     moveList.push(...this.square[i].getLegalMoves(i, this))
364.                 }
365.             }
366.
367.             // All pieces that lie in the attack bitboard of the king in all offsets
368.             const piece = new Piece.Queen(Pieces.white)
369.
370.             const kingBitboard = this.collections[Pieces.king][this.sideToMoveIndex]
371.
372.             let kingPosition = 0n
373.             while (kingBitboard.getBitboard() >> kingPosition !== 1n) {
374.                 kingPosition++
375.             }
376.
377.             const attacks = piece.getAttacks(Number(kingPosition),
378.                 this.collections[Pieces.all][1 - this.sideToMoveIndex].getBitboard())
379.             const isPinning = attacks & this.collections[Pieces.bishop][1 -
380.                 this.sideToMoveIndex].getBitboard()
381.                 | this.collections[Pieces.rook][1 - this.sideToMoveIndex].getBitboard()
382.                 | this.collections[Pieces.queen][1 - this.sideToMoveIndex].getBitboard()
383.
384.             // We only need to check moves of pinned pieces and king moves and en passant

```

```

384.     const sideToPlay = this.sideToMove
385.     const opponentColour = sideToPlay === Pieces.white ? Pieces.black : Pieces.white
386.
387.     const isCheck = this.isSquareAttacked(Number(kingPosition), opponentColour)
388.
389.     moveList = moveList.filter(move => {
390.         const sourceSquare = move.getSourceSquare()
391.
392.         if (this.square[sourceSquare].getType() === Pieces.king) {
393.             this.collections[Pieces.all][this.sideToMoveIndex].remove(sourceSquare)
394.             const isLegal = !this.isSquareAttacked(move.getDestinationSquare(),
opponentColour)
395.             this.collections[Pieces.all][this.sideToMoveIndex].add(sourceSquare)
396.             return isLegal
397.         }
398.
399.         const isPsuedoPinned = isPinning && ((attacks >> BigInt(sourceSquare)) & 1n)
400.         if (isCheck || isPsuedoPinned || move.isEnPassant()) {
401.             this.playMove(move)
402.             const isLegal = !this.isSquareAttacked(Number(kingPosition),
opponentColour)
403.             this.unplayMove(move)
404.
405.             return isLegal
406.         }
407.         else {
408.             return true
409.         }
410.     })
411.
412.     return moveList
413. }
414.
415. isCheck(sideToPlay: number = this.sideToMove) {
416.     const sideToPlayIndex = sideToPlay === Pieces.white ? 0 : 1
417.     const opponentColour = sideToPlay === Pieces.white ? Pieces.black : Pieces.white
418.     const kingBitboard = this.collections[Pieces.king][sideToPlayIndex]
419.
420.     let kingPosition = 0n
421.     while (kingBitboard.getBitboard() >> kingPosition !== 1n) {
422.         kingPosition++
423.     }
424.
425.     return this.isSquareAttacked(Number(kingPosition), opponentColour)
426. }
427.
428. isCheckmate() {
429.     return this.generateLegalMoves().length === 0 && this.isCheck()
430. }
431.
432. isStalemate() {
433.     return this.generateLegalMoves().length === 0 && !this.isCheck()
434. }
435.
436. getBoardData() {
437.     return this.getGameState() >> 1
438. }
439.
440. generateBinaryUCILegalMoves() {
441.     const moveList: Array<Move> = this.generateLegalMoves()
442.     return [...new Set(moveList.map(move => move.toBinaryUCI()))]
443. }
444.
445. toBinary() {
446.     let binaryBoard = new Uint8Array(64)
447.
448.     for (let i = 0; i < 64; i++) {
449.         binaryBoard[i] = this.square[i].datum
450.     }
451.

```

```

452.         return binaryBoard
453.     }
454. }
455.
456. export default Board
457.

```

frontend/src/engine/Move.ts

```

1.  export const Pieces = {
2.    empty: 0,
3.    pawn: 1,
4.    rook: 2,
5.    knight: 3,
6.    bishop: 4,
7.    queen: 5,
8.    king: 6,
9.    black: 8,
10.   white: 16
11. }
12.
13.
14. class Move {
15.   readonly datum: number
16.
17.   private static destinationSquareMask: number = 0xFC00
18.   private static sourceSquareMask: number = 0x03F0
19.   private static flagMask: number = 0x000F
20.
21.   static fromCharacteristics(
22.     dest: number,
23.     source: number,
24.     capture: boolean = false,
25.     doublePawn: boolean = false,
26.     ep: boolean = false,
27.     castle: number = 0, // 1 is ks, 2 is qs
28.     promotion: number = 0 // 1 is knight, 2 is bishop, 3 is rook, and 4 is queen
29.   ) {
30.     const promotionBit = promotion ? 1 : 0
31.     const captureBit = (ep || capture) ? 1 : 0
32.     const special1Bit = (castle || promotion > 2) ? 1 : 0
33.     const special0Bit = (doublePawn || castle == 2 || ep || promotion >> 1 == 1) ? 1 :
0
34.     const flag = promotionBit << 3 | captureBit << 2 | special1Bit << 1 | special0Bit
35.
36.     let datum = (dest << 10) | (source << 4) | flag
37.
38.     return new this(datum)
39.   }
40.
41.
42.   static binarySquareToCoordinate(square: number) {
43.     const rank = Math.floor(square / 8) + 1
44.     const file = String.fromCharCode(97 + (square % 8))
45.
46.     return `${file}${rank}`
47.   }
48.
49.   constructor(datum: number) {
50.     this.datum = datum
51.   }
52.
53.   getDestinationSquare() {
54.     return (this.datum & Move.destinationSquareMask) >> 10
55.   }
56.
57.   getSourceSquare() {
58.     return (this.datum & Move.sourceSquareMask) >> 4

```

```
59.      }
60.
61.      getFlag() {
62.          return this.datum & Move.flagMask
63.      }
64.
65.      toBinaryUCI() {
66.          return this.datum >> 4
67.      }
68.
69.      toLetterUCI() {
70.          const destinationSquare = this.datum >> 10
71.          const sourceSquare = (this.datum & Move.sourceSquareMask) >> 4
72.
73.          let promotionLetter = ""
74.
75.          if (this.isPromotion()) {
76.              switch (this.datum & 0b11) {
77.                  case 0b00:
78.                      promotionLetter = "n"
79.                      break
80.                  case 0b01:
81.                      promotionLetter = "b"
82.                      break
83.                  case 0b10:
84.                      promotionLetter = "r"
85.                      break
86.                  case 0b11:
87.                      promotionLetter = "q"
88.                      break
89.                  default:
90.                      break
91.              }
92.          }
93.
94.          return
` ${Move.binarySquareToCoordinate(sourceSquare)}${Move.binarySquareToCoordinate(destinationSquare)}
)`${promotionLetter}
95.      }
96.
97.      isPromotion() {
98.          return !(this.datum & 0b1000)
99.      }
100.
101.     isCapture() {
102.         return !(this.datum & 0b0100)
103.     }
104.
105.     isEnPassant() {
106.         return (this.datum & Move.flagMask) === 0b0101
107.     }
108.
109.     getPromotionPiece() {
110.         if (!this.isPromotion()) {
111.             return Pieces.empty
112.         }
113.
114.         switch (this.datum & 0b11) {
115.             case 0b00:
116.                 return Pieces.knight
117.             case 0b01:
118.                 return Pieces.bishop
119.             case 0b10:
120.                 return Pieces.rook
121.             case 0b11:
122.                 return Pieces.queen
123.             default:
124.                 return Pieces.empty
125.         }
126.     }
```

```

127. }
128.
129. export default Move
130.

```

frontend/src/engine/Evaluation.ts

```

1. import Board from "./Board";
2. import { Pieces } from "./constants";
3. import { PieceSquareTable, Customisation } from "./Engine";
4.
5. export const pieceValue: { [key: number]: number } = {
6.     0: 0,
7.     1: 100,
8.     2: 500,
9.     3: 300,
10.    4: 300,
11.    5: 900,
12.    6: 0,
13. }
14.
15.
16. export default (board: Board, customisation: Customisation, pieceSquareTables:
PieceSquareTable) => {
17.     // Positive is good for white, negative is good for black
18.
19.     // Material Counting
20.     let whiteMaterial = 0
21.     let whitePieceSquareBonus = 0
22.     let blackMaterial = 0
23.     let blackPieceSquareBonus = 0
24.
25.     // To see if we should start looking for checkmates, we see if the opponent has little
pieces left, by calculating the Hamming weight of the all pieces bitboard
26.     let bitboard = board.getCollection()[Pieces.all][1 -
board.getSideToMoveIndex()].getBitboard()
27.
28.     let hammingWeight = 0
29.
30.     while (bitboard) {
31.         bitboard &= (bitboard - 1n) // Removes the lowest 1s bit
32.         hammingWeight += 1
33.     }
34.
35.     const isOpponentStruggling = hammingWeight < 8n
36.
37.     for (let i = 0; i < 64; i++) {
38.         const pieceType = board.getSquare()[i].getType()
39.         let pieceSquareTableIndex = pieceType === Pieces.King && isOpponentStruggling ?
pieceType + 1 : pieceType
40.
41.         if (board.getSquare()[i].isColour(Pieces.White)) {
42.             whiteMaterial += pieceValue[pieceType]
43.             whitePieceSquareBonus += pieceSquareTables[0][pieceSquareTableIndex][i] *
(isOpponentStruggling ? 0.2 : 1) // Material is more important in the endgame
44.         }
45.         else if ((board.getSquare()[i].isColour(Pieces.Black))) {
46.             blackMaterial += pieceValue[pieceType]
47.             blackPieceSquareBonus += pieceSquareTables[1][pieceSquareTableIndex][i] *
(isOpponentStruggling ? 0.2 : 1)
48.         }
49.     }
50.
51.     // Checkmating - For mates with rooks and queens the king must go towards the sides of
the board and the king must be brought closer
52.     const opponentKingBitboard = board.getCollection()[Pieces.King][1 -
board.getSideToMoveIndex()]
53.     const ourKingBitboard = board.getCollection()[Pieces.King][board.getSideToMoveIndex()]

```

```

54.
55.     let checkmateBonus = 0
56.
57.     if (isOpponentStruggling) {
58.         let opponentKingPositionBig = 0n
59.         while (opponentKingBitboard.getBitboard() >> opponentKingPositionBig !== 1n) {
60.             opponentKingPositionBig++
61.         }
62.
63.         let ourKingPositionBig = 0n
64.         while (ourKingBitboard.getBitboard() >> ourKingPositionBig !== 1n) {
65.             ourKingPositionBig++
66.         }
67.
68.         const opponentKingPosition = Number(opponentKingPositionBig)
69.         const ourKingPosition = Number(ourKingPosition)
70.
71.         // Taxicab distance between kings
72.         const kingDistance = Math.abs((opponentKingPosition % 8) - (ourKingPosition % 8)) +
Math.abs(Math.floor(opponentKingPosition / 8) - Math.floor(ourKingPosition / 8))
73.
74.         checkmateBonus += 10 * (16 - kingDistance)
75.
76.         // Taxicab distance between opponent king and corner of the board
77.         const corneringDistance = ((opponentKingPosition % 8) % 7) +
(Math.floor(opponentKingPosition / 8) % 7)
78.
79.         checkmateBonus += 25 * (16 - corneringDistance)
80.     }
81.
82.     // If happy to trade, then we prefer boards with fewer pieces,
83.     if (board.getSideToMove() === Pieces.white) {
84.         return whiteMaterial - blackMaterial + whitePieceSquareBonus - blackPieceSquareBonus
+ checkmateBonus + (-(customisation.tradeHappy - 50) * blackMaterial) * 0.0005
85.     }
86.     else {
87.         return blackMaterial - whiteMaterial + blackPieceSquareBonus - whitePieceSquareBonus
+ checkmateBonus + (-(customisation.tradeHappy - 50) * whiteMaterial) * 0.0005
88.     }
89.
90. }
91.

```

frontend/src/engine/Pieces/index.ts

```

1. import Pawn from "./Pawn";
2. import Queen from "./Queen";
3. import Rook from "./Rook";
4. import Empty from "./Empty";
5. import { Pieces } from "../constants";
6. import BasePiece from "./BasePiece";
7.
8. function getPieceFromBinary(datum: number): BasePiece {
9.     const colour = datum & 0b1100
10.
11.    switch (datum & 0b00111) {
12.        case Pieces.pawn:
13.            return new Pawn(colour)
14.        case Pieces.rook:
15.            return new Rook(colour)
16.        case Pieces.knight:
17.            return new Knight(colour)
18.        case Pieces.bishop:
19.            return new Bishop(colour)
20.        case Pieces.queen:
21.            return new Queen(colour)
22.        case Pieces.king:
23.            return new King(colour)

```

```

24.     default:
25.         return new Empty()
26.     }
27.
28. }
29.
30.
31. export default {
32.     Bishop,
33.     King,
34.     Knight,
35.     Pawn,
36.     Queen,
37.     Rook,
38.     Empty,
39.     FromBinary: getPieceFromBinary
40. };
41.

```

frontend/src/engine/Pieces/Rook.ts

```

1. import BasePiece from "./BasePiece";
2. import { Pieces } from "../constants";
3. import Board from "../Board";
4. import { generateSliderMoves, getSlidingPieceAttacks } from "./utils";
5. import Move from "../Move";
6.
7.
8. class Rook extends BasePiece {
9.     constructor(colour: number) {
10.         super(colour | Pieces.rook)
11.     }
12.
13.     public override getAttacks(square: number, blockers: bigint) {
14.         return getSlidingPieceAttacks(square, [-1, 8, 1, -8], blockers)
15.     }
16.
17.     public override getLegalMoves(square: number, board: Board): Array<Move> {
18.         return generateSliderMoves(square, this.getColour(), [-1, 8, 1, -8], board)
19.     }
20.
21. }
22.
23. export default Rook
24.

```

frontend/src/engine/Pieces/Queen.ts

```

1. import BasePiece from "./BasePiece";
2. import { Pieces } from "../constants";
3. import Board from "../Board";
4. import { generateSliderMoves, getSlidingPieceAttacks } from "./utils";
5. import Move from "../Move";
6.
7.
8. class Queen extends BasePiece {
9.     constructor(colour: number) {
10.         super(colour | Pieces.queen)
11.     }
12.
13.     public override getAttacks(square: number, blockers: bigint) {
14.         return getSlidingPieceAttacks(square, [-1, 7, 8, 9, 1, -7, -8, -9], blockers)
15.     }
16.
17.     public override getLegalMoves(square: number, board: Board): Array<Move> {
18.

```

```

19.         return generateSliderMoves(square, this.getColour(), [-1, 7, 8, 9, 1, -7, -8, -9],
20.     }
21. }
22.
23. export default Queen
24.
```

frontend/src/engine/Pieces/Pawn.ts

```

1. import BasePiece from "./BasePiece";
2. import { Pieces } from "../constants";
3. import Board from "../Board";
4. import Move from "../Move";
5. import SquareCollection from "../SquareCollection";
6.
7.
8. class Pawn extends BasePiece {
9.     constructor(colour: number) {
10.         super(colour | Pieces.pawn)
11.     }
12.
13.     public override getAttacks(square: number, blockers: bigint) {
14.         const pieceColour = this.getColour()
15.
16.         let attackBitboard = 0n
17.
18.         const movementDirection = pieceColour == Pieces.white ? 1 : -1
19.         const file = square % 8
20.
21.         const LHCaptureSquare: number = square + movementDirection * 7
22.         const RHCaptureSquare: number = square + movementDirection * 9
23.
24.         const isOnLeftFile: boolean = file === (pieceColour == Pieces.white ? 0 : 7)
25.         const isOnRightFile: boolean = file === (pieceColour == Pieces.white ? 7 : 0)
26.
27.         if (!isOnLeftFile) {
28.             attackBitboard |= 1n << BigInt(LHCaptureSquare)
29.         }
30.
31.         if (!isOnRightFile) {
32.             attackBitboard |= 1n << BigInt(RHCaptureSquare)
33.         }
34.
35.         return attackBitboard
36.
37.     }
38.
39.     public override getLegalMoves(square: number, board: Board): Array<Move> {
40.         const pieceColour = this.getColour()
41.
42.         let moves: Array<Move> = []
43.
44.         // If we're white, we're going up the board, if we're black we need to go down
45.         const movementDirection = pieceColour == Pieces.white ? 1 : -1
46.         const pieceColourIndex = pieceColour == Pieces.white ? 0 : 1
47.
48.         const rank = Math.floor(square / 8)
49.         const file = square % 8
50.
51.         // One rank ahead, given the target square isn't blocked
52.         const squareAhead = square + movementDirection * 8
53.
54.         const promotionRank = pieceColour == Pieces.white ? 6 : 1
55.
56.         if (board.getSquares()[squareAhead].getType() == Pieces.empty) {
57.             if (rank === promotionRank) {
58.                 for (let i = 1; i < 5; i++) {
```

```

59.             moves.push(Move.fromCharacteristics(squareAhead, square, false, false,
false, 0, i))
60.         }
61.     }
62.   else {
63.     moves.push(Move.fromCharacteristics(squareAhead, square))
64.     // Two ranks ahead, given the target square isn't blocked
65.     const isFirstMove = ((pieceColour == Pieces.white) && (rank == 1)) ||
((pieceColour == Pieces.black) && (rank == 6))
66.
67.     if (isFirstMove) {
68.       const squareTwoAhead = squareAhead + movementDirection * 8
69.
70.       if (board.getSquares()[squareTwoAhead].getType() == Pieces.empty) {
71.         moves.push(Move.fromCharacteristics(squareTwoAhead, square, false,
true))
72.       }
73.     }
74.   }
75.
76. }
77.
78. // En passant
79. if (board.getEpFile() !== 0) {
80.   const LHCaptureSquare: number = square + movementDirection * 7
81.   const RHCaptureSquare: number = square + movementDirection * 9
82.   const epRank = pieceColour === Pieces.white ? 4 : 3
83.   const epFile = board.getEpFile() - 1
84.
85.   if (epRank === rank && (epFile - 1 === file || epFile + 1 === file)) {
86.     const enPassantCaptureSquare = movementDirection * epFile >
movementDirection * file ? RHCaptureSquare : LHCaptureSquare
87.     moves.push(Move.fromCharacteristics(enPassantCaptureSquare, square, true,
false, true))
88.   }
89. }
90.
91.
92. // Captures and promotions
93. const attacks = new SquareCollection(this.getAttacks(square, 0n))
94.
95. // Bitwise AND on the attacks and the opponent's pieces to see which attacks are
actually captures
96. const captures = attacks.and(board.getCollections()[Pieces.all][1 -
pieceColourIndex])
97.
98. for (const captureSquare of captures) {
99.   if (rank === promotionRank) {
100.     for (let i = 1; i < 5; i++) {
101.       moves.push(Move.fromCharacteristics(captureSquare, square, true, false,
false, 0, i))
102.     }
103.   } else {
104.     moves.push(Move.fromCharacteristics(captureSquare, square, true))
105.   }
106. }
107.
108.
109. return moves
110. }
111. }
112.
113. export default Pawn
114.

```

frontend/src/engine/Pieces/Knight.ts

```
1. import BasePiece from "./BasePiece";
```

```

2. import { Pieces } from "../constants";
3. import Board from "../Board";
4. import Move from "../Move";
5. import { precomputedKnightMoves } from "./utils/precalculations/results";
6. import SquareCollection from "../SquareCollection";
7.
8.
9. class Knight extends BasePiece {
10.   constructor(colour: number) {
11.     super(colour | Pieces.knight)
12.   }
13.
14.   public override getAttacks(square: number, blockers: bigint) {
15.     const squaresInRange = precomputedKnightMoves.get(square)
16.
17.     if (!squaresInRange) {
18.       throw 'Error using precomputed knight data'
19.     }
20.
21.     let attackBitboard = 0n
22.
23.
24.     for (const destSquare of squaresInRange) {
25.       attackBitboard |= 1n << BigInt(destSquare)
26.     }
27.
28.     return attackBitboard
29.   }
30.
31.   public override getLegalMoves(square: number, board: Board): Array<Move> {
32.     let moves: Array<Move> = []
33.
34.     const pieceColour = this.getColour()
35.
36.     const attacks = new SquareCollection(this.getAttacks(square, 0n))
37.
38.     const pieceColourIndex = pieceColour == Pieces.white ? 0 : 1
39.
40.     const opponentPieces = board.getCollections()[Pieces.all][1 - pieceColourIndex]
41.     const friendlyPieces = board.getCollections()[Pieces.all][pieceColourIndex]
42.
43.     // Captures
44.     const captures = attacks.and(opponentPieces)
45.
46.     for (const captureSquare of captures) {
47.       moves.push(Move.fromCharacteristics(captureSquare, square, true))
48.     }
49.
50.     // Quiet Moves
51.     const quietMoves = attacks.and(opponentPieces.or(friendlyPieces).not())
52.
53.     for (const quietMove of quietMoves) {
54.       moves.push(Move.fromCharacteristics(quietMove, square))
55.     }
56.
57.     return moves
58.   }
59. }
60.
61. export default Knight
62.

```

frontend/src/engine/Pieces/King.ts

```

1. import BasePiece from "./BasePiece";
2. import { Pieces } from "../constants";
3. import Board from "../Board";
4. import { generateSliderMoves, getSlidingPieceAttacks } from "./utils";

```

```

5. import Move from "../Move";
6.
7.
8. class King extends BasePiece {
9.     constructor(colour: number) {
10.         super(colour | Pieces.king)
11.     }
12.
13.     public override getAttacks(square: number, blockers: bigint) {
14.         // King cannot be blocked (as it has distance 1), so the empty bitboard of 0 will
work
15.         return getSlidingPieceAttacks(square, [-1, 7, 8, 9, 1, -7, -8, -9], 0n, 1)
16.     }
17.
18.     public override getLegalMoves(square: number, board: Board): Array<Move> {
19.         const pieceColour = this.datum & 0b11000
20.
21.         // Normal king moves can be thought of a sliding piece with distance 1
22.         let moves: Array<Move> = generateSliderMoves(square, pieceColour, [-1, 7, 8, 9, 1, -7, -8, -9], board, 1)
23.
24.         // Castling - MSB is Queen-side castling, LSB is King-side castling. 1 is
unavailable, 0 is available.
25.         let castlingRights: number
26.
27.         const opponentColour = pieceColour == Pieces.white ? Pieces.black : Pieces.white
28.
29.         if (pieceColour === Pieces.white) {
30.             castlingRights = board.getCastlingRights() & 0b0011
31.         }
32.         else {
33.             castlingRights = (board.getCastlingRights() & 0b1100) >> 2
34.         }
35.
36.         // King-side Castling
37.         if ((castlingRights & 0b01) === 0) {
38.             const indexesBetween = pieceColour === Pieces.white ? [5, 6] : [61, 62]
39.             const isLegal = indexesBetween.every(i => board.getSquares()[i].getType() ===
Pieces.empty && !board.isSquareAttacked(i, opponentColour))
40.
41.             if (isLegal && !board.isSquareAttacked(square, opponentColour)) {
42.                 moves.push(Move.fromCharacteristics(square + 2, square, false, false, false,
1))
43.             }
44.         }
45.
46.         // Queen-side Castling
47.         if ((castlingRights & 0b10) === 0) {
48.             const indexesBetween = pieceColour === Pieces.white ? [2, 3] : [58, 59]
49.             const squareRookMovesThrough = pieceColour === Pieces.white ? 1 : 57
50.
51.             const isLegal = indexesBetween.every(i => {
52.                 return board.getSquares()[i].getType() === Pieces.empty &&
!board.isSquareAttacked(i, opponentColour)
53.             })
54.
55.             if (isLegal && !board.isSquareAttacked(square, opponentColour) &&
board.getSquares()[squareRookMovesThrough].getType() === Pieces.empty) {
56.                 moves.push(Move.fromCharacteristics(square - 2, square, false, false, false,
2))
57.             }
58.         }
59.
60.         return moves
61.     }
62. }
63.
64. export default King
65.

```

frontend/src/engine/Pieces/Bishop.ts

```

1. import BasePiece from "./BasePiece";
2. import { Pieces } from "../constants";
3. import Board from "../Board";
4. import { generateSliderMoves, getSlidingPieceAttacks } from "./utils";
5. import Move from "../Move";
6.
7.
8. class Bishop extends BasePiece {
9.     constructor(colour: number) {
10.         super(colour | Pieces.bishop)
11.     }
12.
13.     public override getAttacks(square: number, blockers: bigint) {
14.         return getSlidingPieceAttacks(square, [7, 9, -7, -9], blockers)
15.     }
16.
17.     public override getLegalMoves(square: number, board: Board): Array<Move> {
18.         return generateSliderMoves(square, this.getColour(), [7, 9, -7, -9], board)
19.     }
20. }
21.
22. export default Bishop
23.
```

frontend/src/engine/Pieces/Empty.ts

```

1. import BasePiece from "./BasePiece";
2.
3. class Empty extends BasePiece {
4.     constructor() {
5.         super(0)
6.     }
7. }
8.
9. export default Empty
10.
```

frontend/src/engine/Pieces/BasePiece.ts

```

1. import Board from '../Board';
2. import Move from '../Move';
3.
4. // Abstract base class representing a Piece
5. class BasePiece {
6.     readonly datum: number
7.
8.     constructor(datum: number) {
9.         if (this.constructor === BasePiece) {
10.             throw new Error("Cannot construct BasePiece abstract class")
11.         }
12.
13.         this.datum = datum
14.     }
15.
16.     public getColour() {
17.         return this.datum & 0b11000
18.     }
19.
20.     public getType() {
21.         return this.datum & 0b00111
22.     }
23.
24.     public isColour(colour: number) {
```

```

25.         return (this.datum & 0b11000) === colour
26.     }
27.
28.     public getColourIndex() {
29.         // 0 is white 1 is black
30.         return +!!(this.datum & 8)
31.     }
32.
33.     public getOpponentColourIndex() {
34.         return +!(this.datum & 8)
35.     }
36.
37.     public getLegalMoves(square: number, board: Board): Array<Move> {
38.         return []
39.     }
40.
41.     public getAttacks(square: number, blockers: bigint): bigint {
42.         return 0n
43.     }
44.
45.     public isDifferentColour(piece: BasePiece) {
46.         // Shift datum so only the white bit is left
47.         // If the pieces are different the XOR operation will be true
48.         return (piece.datum >> 4) ^ (this.datum >> 4)
49.     }
50.
51. }
52.
53. export default BasePiece
54.

```

frontend/src/engine/Pieces/utils/index.ts

```

1. import Board from "../../Board"
2. import Move from "../../Move"
3. import SquareCollection from "../../SquareCollection"
4. import { Pieces } from "../../constants"
5. import { precomputedSlidersSerialisedDistances } from "./precalculations/results"
6.
7. export function getSlidingPieceAttacks(square: number, offsets: Array<number>, blockers: bigint, distance: number = 8) {
8.     const allOffsets = [-1, 7, 8, 9, 1, -7, -8, -9]
9.
10.    let distancesFromEdge = precomputedSlidersSerialisedDistances.get(square)
11.
12.    if (!distancesFromEdge) {
13.        throw 'Error using precomputed slider data'
14.    }
15.
16.    let attackBitboard = 0n
17.
18.    distancesFromEdge = distancesFromEdge.filter((_, i) => offsets.includes(allOffsets[i]))
19.
20.    for (let i = 0; i < offsets.length; i++) {
21.        for (let j = 0; j < Math.min(distancesFromEdge[i], distance); j++) {
22.            const destSquareBitboardValue = 1n << BigInt((offsets[i] * (j + 1)) + square)
23.
24.            attackBitboard |= destSquareBitboardValue
25.
26.            // If we've run into a piece, we stop counting attacks on this offset
27.            if (blockers & destSquareBitboardValue) {
28.                break
29.            }
30.        }
31.    }
32.
33.    return attackBitboard
34. }

```

```

35.
36. export function generateSliderMoves(square: number, pieceColour: number, offsets: Array<number>, board: Board, distance: number = 8) {
37.     const pieceColourIndex = pieceColour == Pieces.white ? 0 : 1
38.
39.     const opponentPieces = board.getCollection(Pieces.all)[1 - pieceColourIndex]
40.     const friendlyPieces = board.getCollection(Pieces.all)[pieceColourIndex]
41.
42.     const attacks = getSlidingPieceAttacks(square, offsets,
43.         opponentPieces.or(friendlyPieces).getBitboard(), distance)
43.
44.     let moves: Array<Move> = []
45.
46.     // Captures
47.     const captures = attacks & opponentPieces.getBitboard()
48.
49.     for (const captureSquare of new SquareCollection(captures)) {
50.         moves.push(Move.fromCharacteristics(captureSquare, square, true))
51.     }
52.
53.     // Quiet Moves
54.     const quietMoves = attacks & (opponentPieces.or(friendlyPieces).not()).getBitboard()
55.
56.     for (const quietMove of new SquareCollection(quietMoves)) {
57.         moves.push(Move.fromCharacteristics(quietMove, square))
58.     }
59.
60.     return moves
61. }
62.

```

frontend/src/engine/Pieces/utils/precalculations/knightMoves.js

```

1. // Knight offsets
2. const offsets = [
3.     { offset: -10, disallowedFiles: [0, 1], disallowedRanks: [0] },
4.     { offset: -17, disallowedFiles: [0], disallowedRanks: [0, 1] },
5.     { offset: -15, disallowedFiles: [7], disallowedRanks: [0, 1] },
6.     { offset: -6, disallowedFiles: [6, 7], disallowedRanks: [0] },
7.     { offset: 6, disallowedFiles: [0, 1], disallowedRanks: [7] },
8.     { offset: 15, disallowedFiles: [0], disallowedRanks: [6, 7] },
9.     { offset: 17, disallowedFiles: [7], disallowedRanks: [6, 7] },
10.    { offset: 10, disallowedFiles: [6, 7], disallowedRanks: [7] },
11. ]
12.
13. const moves = new Map()
14.
15. for (let i = 0; i < 64; i++) {
16.     const file = i % 8
17.     const rank = Math.floor(i / 8)
18.
19.     const movesFromSquare = []
20.
21.     for (let j = 0; j < offsets.length; j++) {
22.         if (offsets[j].disallowedFiles.includes(file) ||
23.             offsets[j].disallowedRanks.includes(rank)) {
24.             continue
25.         }
26.
27.         const targetSquare = i + offsets[j].offset
28.
29.         movesFromSquare.push(targetSquare)
30.     }
31.
32.     moves.set(i, movesFromSquare)
33. }
34.

```

```

35. const serialisedMoves = JSON.stringify(Array.from(moves.entries()))
36.
37. console.log(serialisedMoves)
38.

```

frontend/src/engine/Pieces/utils/precalculations/slidingPiecesMoves.js

```

1. const distancesFromEdge = new Map()
2.
3. for (let i = 0; i < 64; i++) {
4.   const file = i % 8
5.   const rank = Math.floor(i / 8)
6.
7.   let distanceFromTop = 7 - rank
8.   let distanceFromBottom = rank
9.   let distanceFromLeft = file
10.  let distanceFromRight = 7 - file
11.
12.  // Corresponding to these offsets [-1, 7, 8, 9, 1, -7, -8, -9]
13.  distancesFromEdge.set(i, [
14.    distanceFromLeft,
15.    Math.min(distanceFromLeft, distanceFromTop),
16.    distanceFromTop,
17.    Math.min(distanceFromTop, distanceFromRight),
18.    distanceFromRight,
19.    Math.min(distanceFromBottom, distanceFromRight),
20.    distanceFromBottom,
21.    Math.min(distanceFromBottom, distanceFromLeft)
22.  ])
23. }
24.
25. const serialisedDistances = JSON.stringify(Array.from(distancesFromEdge.entries()))
26.
27. console.log(serialisedDistances)
28.

```

frontend/src/engine/Pieces/utils/precalculations/results.ts

```

1. const knightSerialisedMoves =
"[[0,[17,10]],[1,[16,18,11]],[2,[8,17,19,12]],[3,[9,18,20,13]],[4,[10,19,21,14]],[5,[11,20,22,15]
]], [6,[12,21,23]],[7,[13,22]],[8,[2,25,18]],[9,[3,24,26,19]],[10,[0,4,16,25,27,20]],[11,[1,5,17,
26,28,21]],[12,[2,6,18,27,29,22]],[13,[3,7,19,28,30,23]],[14,[4,20,29,31]],[15,[5,21,30]],[16,[1
,10,33,26]],[17,[0,2,11,32,34,27]],[18,[8,1,3,12,24,33,35,28]],[19,[9,2,4,13,25,34,36,29]],[20,[
10,3,5,14,26,35,37,30]],[21,[11,4,6,15,27,36,38,31]],[22,[12,5,7,28,37,39]],[23,[13,6,29,38]],[2
4,[9,18,41,34]],[25,[8,10,19,40,42,35]],[26,[16,9,11,20,32,41,43,36]],[27,[17,10,12,21,33,42,44,
37]],[28,[18,11,13,22,34,43,45,38]],[29,[19,12,14,23,35,44,46,39]],[30,[20,13,15,36,45,47]],[31,
[21,14,37,46]],[32,[17,26,49,42]],[33,[16,18,27,48,50,43]],[34,[24,17,19,28,40,49,51,44]],[35,[2
5,18,20,29,41,50,52,45]],[36,[26,19,21,30,42,51,53,46]],[37,[27,20,22,31,43,52,54,47]] [38,[28,2
1,23,44,53,55]],[39,[29,22,45,54]],[40,[25,34,57,50]],[41,[24,26,35,56,58,51]],[42,[32,25,27,36,
48,57,59,52]],[43,[33,26,28,37,49,58,60,53]],[44,[34,27,29,38,50,59,61,54]],[45,[35,28,30,39,51,
60,62,55]],[46,[36,29,31,52,61,63]],[47,[37,30,53,62]],[48,[33,42,58]],[49,[32,34,43,59]],[50,[4
0,33,35,44,56,60]],[51,[41,34,36,45,57,61]],[52,[42,35,37,46,58,62]],[53,[43,36,38,47,59,63]],[5
4,[44,37,39,60]],[55,[45,38,61]],[56,[41,50]],[57,[40,42,51]],[58,[48,41,43,52]],[59,[49,42,44,5
3]],[60,[50,43,45,54]],[61,[51,44,46,55]],[62,[52,45,47]],[63,[53,46]]]"
2. const slidersSerialisedDistances =
"[[0,[0,0,7,7,7,0,0,0]],[1,[1,1,7,6,6,0,0,0]],[2,[2,2,7,5,5,0,0,0]],[3,[3,3,7,4,4,0,0,0]],[4,[4,
4,7,3,3,0,0,0]],[5,[5,5,7,2,2,0,0,0]],[6,[6,6,7,1,1,0,0,0]],[7,[7,7,7,0,0,0,0,0]],[8,[0,0,6,6,7,
1,1,0]],[9,[1,1,6,6,6,1,1,1]],[10,[2,2,6,5,5,1,1,1]],[11,[3,3,6,4,4,1,1,1]],[12,[4,4,6,3,3,1,1,1
]],[13,[5,5,6,2,2,1,1,1]],[14,[6,6,6,1,1,1,1,1]],[15,[7,6,6,0,0,0,1,1]],[16,[0,0,5,5,7,2,2,0]],[1
7,[1,1,5,5,6,2,2,1]]],[18,[2,2,5,5,5,2,2,2]],[19,[3,3,5,4,4,2,2,2]],[20,[4,4,5,3,3,2,2,2]],[21,[
5,5,5,2,2,2,2,2]],[22,[6,5,5,1,1,1,2,2]],[23,[7,5,5,0,0,0,2,2]],[24,[0,0,4,4,7,3,3,0]],[25,[1,1,
4,4,6,3,3,1]],[26,[2,2,4,4,5,3,3,2]],[27,[3,3,4,4,4,3,3,3]],[28,[4,4,4,3,3,3,3,3]],[29,[5,4,4,2,
2,2,3,3]],[30,[6,4,4,1,1,1,3,3]],[31,[7,4,4,0,0,0,3,3]],[32,[0,0,3,3,7,4,4,0]],[33,[1,1,3,3,6,4,
4,1]],[34,[2,2,3,3,5,4,4,2]],[35,[3,3,3,3,4,4,4,3]],[36,[4,3,3,3,3,3,4,4,4]],[37,[5,3,3,2,2,2,4,4
]],[38,[6,3,3,1,1,1,4,4]],[39,[7,3,3,0,0,0,4,4]],[40,[0,0,2,2,7,5,5,0]],[41,[1,1,2,2,6,5,5,1]],[4
2,[2,2,2,2,5,5,5,2]],[43,[3,2,2,2,4,4,5,3]],[44,[4,2,2,2,3,3,5,4]],[45,[5,2,2,2,2,2,5,5]],[46,[6

```

```

1. ,2,2,1,1,1,5,5]], [47,[7,2,2,0,0,0,5,5]], [48,[0,0,1,1,7,6,6,0]], [49,[1,1,1,1,6,6,6,1]], [50,[2,1,1
2,1,5,5,6,2]], [51,[3,1,1,1,4,4,6,3]], [52,[4,1,1,1,3,3,6,4]], [53,[5,1,1,1,2,2,6,5]], [54,[6,1,1,1,1
,1,6,6]], [55,[7,1,1,0,0,0,6,6]], [56,[0,0,0,0,7,7,7,0]], [57,[1,0,0,0,6,6,7,1]], [58,[2,0,0,0,5,5,7
,2]], [59,[3,0,0,0,4,4,7,3]], [60,[4,0,0,0,3,3,7,4]], [61,[5,0,0,0,2,2,7,5]], [62,[6,0,0,0,1,1,7,6]]
,[63,[7,0,0,0,0,0,7,7]]]"  

3.  

4. export const precomputedKnightMoves: Map<number, Array<number>> = new  

Map(JSON.parse(knightSerialisedMoves))  

5. export const precomputedSlidersSerialisedDistances: Map<number, Array<number>> = new  

Map(JSON.parse(slidersSerialisedDistances))  

6.

```

frontend/src/components/BoardElement/constants.tsx

```

1. interface BoardCustomisation {  

2.   lightSquares: string  

3.   darkSquares: string  

4.   activeSquare: string  

5.   allowedMove: string  

6.   lastMoveDestination: string  

7.   lastMoveSource: string  

8. }  

9.  

10. export const squareLength: number = 100 // Pixels  

11.  

12. export const initialColours: BoardCustomisation = {  

13.   lightSquares: "#f0d9b5",  

14.   darkSquares: "#b58863",  

15.   activeSquare: "#ffffff",  

16.   allowedMove: "#000",  

17.   lastMoveDestination: "#ffd47",  

18.   lastMoveSource: "#ffeb94"  

19. }  

20.  

21. export const Pieces = {  

22.   empty: 0,  

23.   pawn: 1,  

24.   rook: 2,  

25.   knight: 3,  

26.   bishop: 4,  

27.   queen: 5,  

28.   king: 6,  

29.   black: 8,  

30.   white: 16  

31. }  

32.

```

frontend/src/components/BoardElement/pieces.svg

[Intentionally Omitted]

frontend/src/components/BoardElement/index.css

```

1. /* Chess board */  

2.  

3. .board {  

4.   position: relative;  

5. }  

6.  

7. .squares,  

8. .square-masks {  

9.   display: grid;  

10.  grid-template-columns: repeat(8, 1fr);  

11.  grid-template-rows: repeat(8, 1fr);  

12. }

```

```

13.
14. .square-masks {
15.   z-index: 2;
16.   position: absolute;
17.   opacity: 0.5;
18.   top: 0;
19.   left: 0;
20. }
21.
22.
23. .chess-piece {
24.   position: absolute;
25.   left: 0;
26.   cursor: pointer;
27. }

```

frontend/src/components/Piece.ts

```

1. import React from "react"
2. import { Pieces } from "./constants"
3. import pieceSVGs from "./pieces.svg"
4.
5.
6. interface PieceProps {
7.   piece: number;
8.   style?: React.CSSProperties;
9.   onMouseDown?: (event: React.MouseEvent) => void;
10.  onMouseMove?: (event: React.MouseEvent) => void;
11.  onMouseUp?: () => void;
12. }
13.
14.
15. class Piece extends React.Component<PieceProps> {
16.   constructor(props: PieceProps) {
17.     super(props)
18.   }
19.
20.   render() {
21.     const { piece, style, onMouseDown, onMouseMove, onMouseUp } = this.props
22.
23.     const colour: string = piece & Pieces.white ? "white" : "black"
24.
25.     let type: string = ""
26.
27.     switch (piece & 7) {
28.       case Pieces.pawn:
29.         type = "pawn"
30.         break
31.       case Pieces.rook:
32.         type = "rook"
33.         break
34.       case Pieces.knight:
35.         type = "knight"
36.         break
37.       case Pieces.bishop:
38.         type = "bishop"
39.         break
40.       case Pieces.queen:
41.         type = "queen"
42.         break
43.       case Pieces.king:
44.         type = "king"
45.         break
46.     }
47.
48.     return (
49.       <svg className="chess-piece" style={style} onMouseDown={onMouseDown}
| onMouseUp={onMouseUp} onMouseMove={onMouseMove}>

```