General Overview

This document is an attempt to codify chess using cartesian coordinates and mathematics. This document should be good enough for any software engineer to pick it up, read it and be able to implement chess as an application in any programming language.

It will also serve as a programming language learning project. To learn multiple language syntax and compare key strengths of multiple programming languages. The entire moves algorithm, system architecture and requirements, as well as UML diagram will be found in this manual.

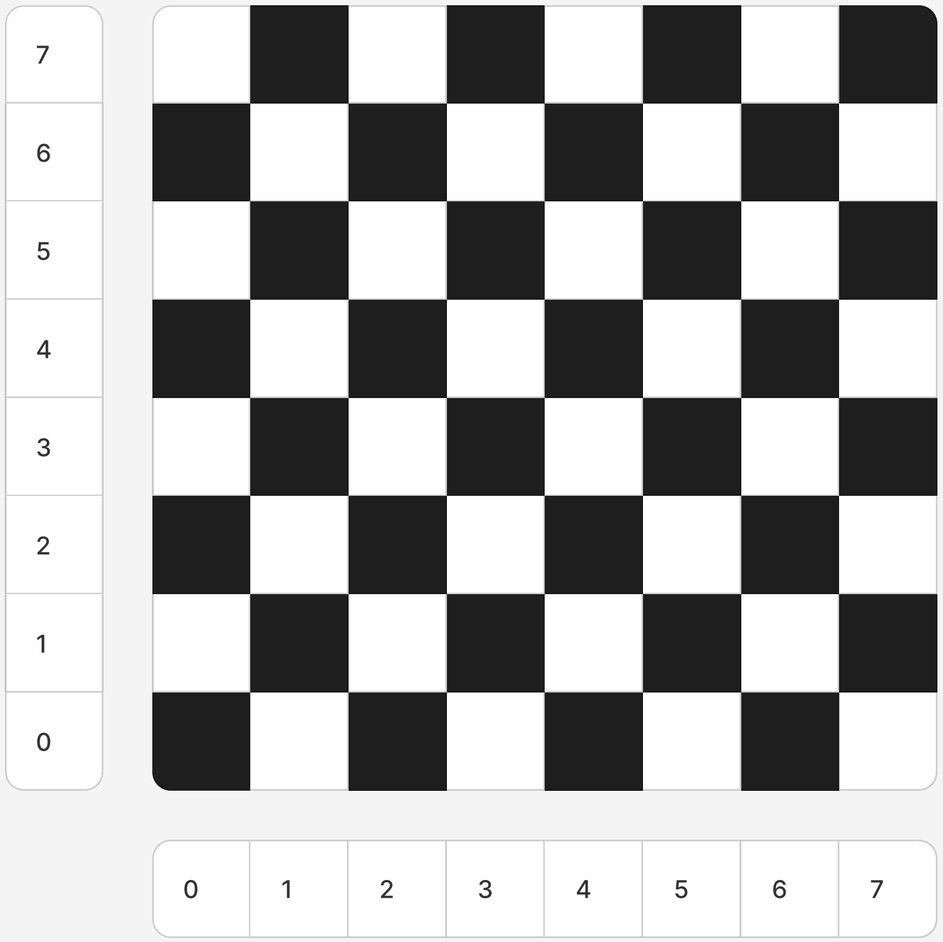
System Requirements

This is a multiplayer chess game that accepts users over the internet. It can show ELO ratings of players on request. It matches players randomly based on available players in a queue. The system should be able to handle over 10000 play requests per second, and match the players appropriately. In terms of availability, four 9s can be adopted as the play experience needs to be excellent for users to continue using it. Fault tolerance for this system should be 100% as users cannot afford to lose games during play. The game should have both a guest mode and user mode giving users the ability to choose whether to login with a specific email address and password or choose just a username and password. User usage analytics and game history needs to be saved and can be reviewed persistently by the user. Users can chat with each other and organize tournaments amongst themselves but won’t have the ability to add each other as friends.

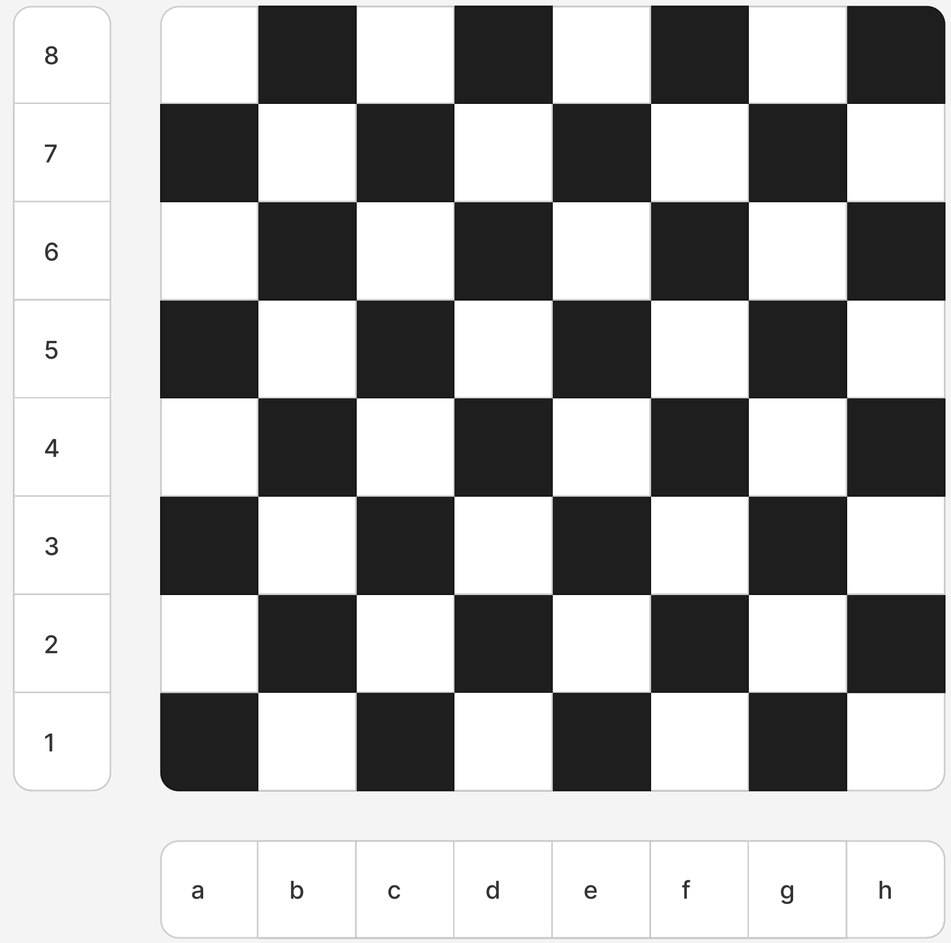
Move Algorithms

Board Representation

Cartesian Representation of a chess board starting at origin



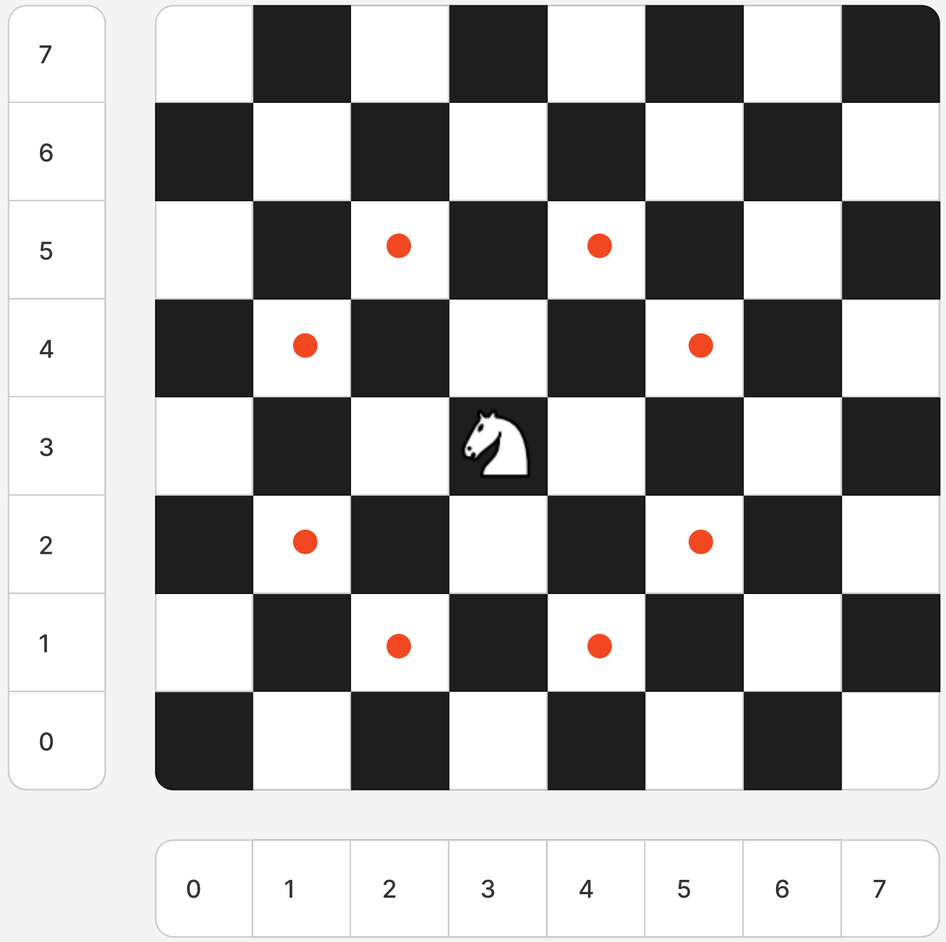
Algebraic Representation of a chess board using standard notation



Using Cartesian Coordinates, we can represent an 8x8 chessboard as seen above. Mapping that with the standard algebraic notation of chess, the d3 square is represented by (3,2) in cartesian coordinates.

This way, we can calculate the difference between current piece square and proposed piece square.

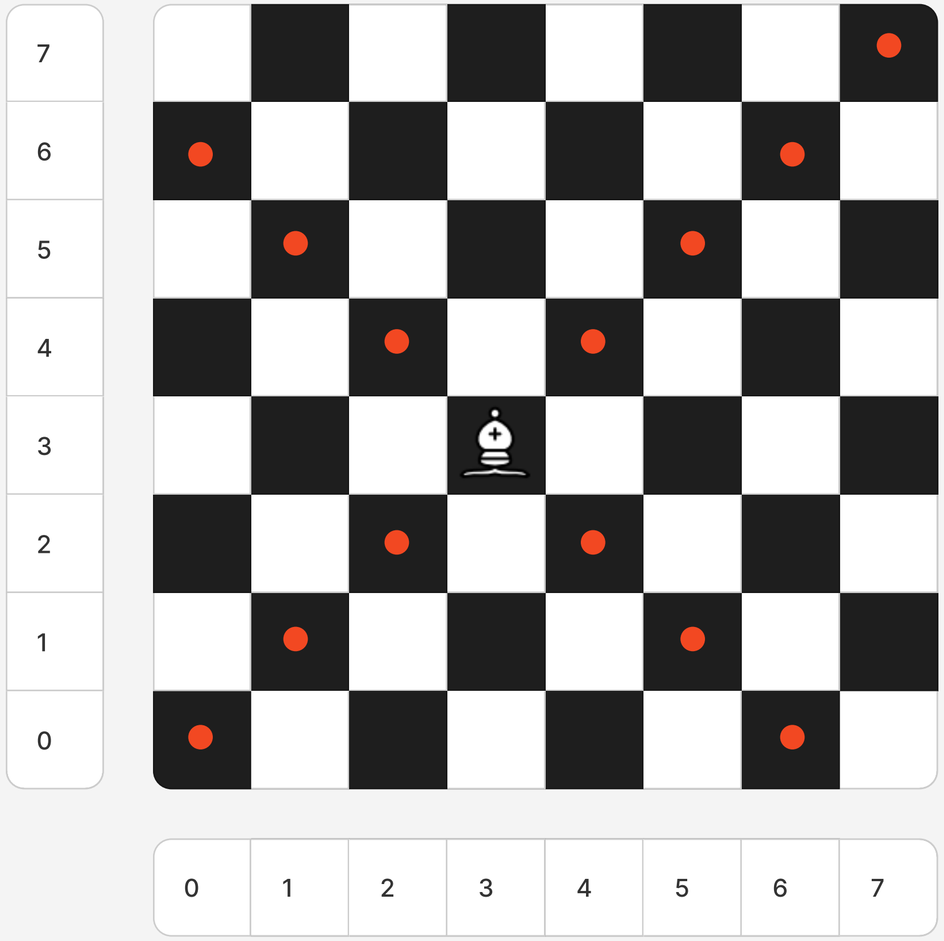
Knights




```
Say we have a knight in the center of the board on square (3,3), the possible squares a knight is
legally allowed to move to are listed below:
1. (2,1) and (1,2)
2. (4,1) and (1,4)
3. (5,2) and (2,5)
4. (5,4) and (4,5)

Difference between the current position (x1,y1) and the possible positions (x2,y2). (x1-x2)/(y1-y2)
1. (2,1) and (1,2) => 0.5 and 2
2. (4,1) and (1,4) => -0.5 and -2
3. (5,2) and (2,5) => -2 and -0.5
4. (5,4) and (4,5) => -2 and  0.5
As seen above the for a knight to make a valid move, the diff is either -2, 2, -0.5 or 0.5.
```

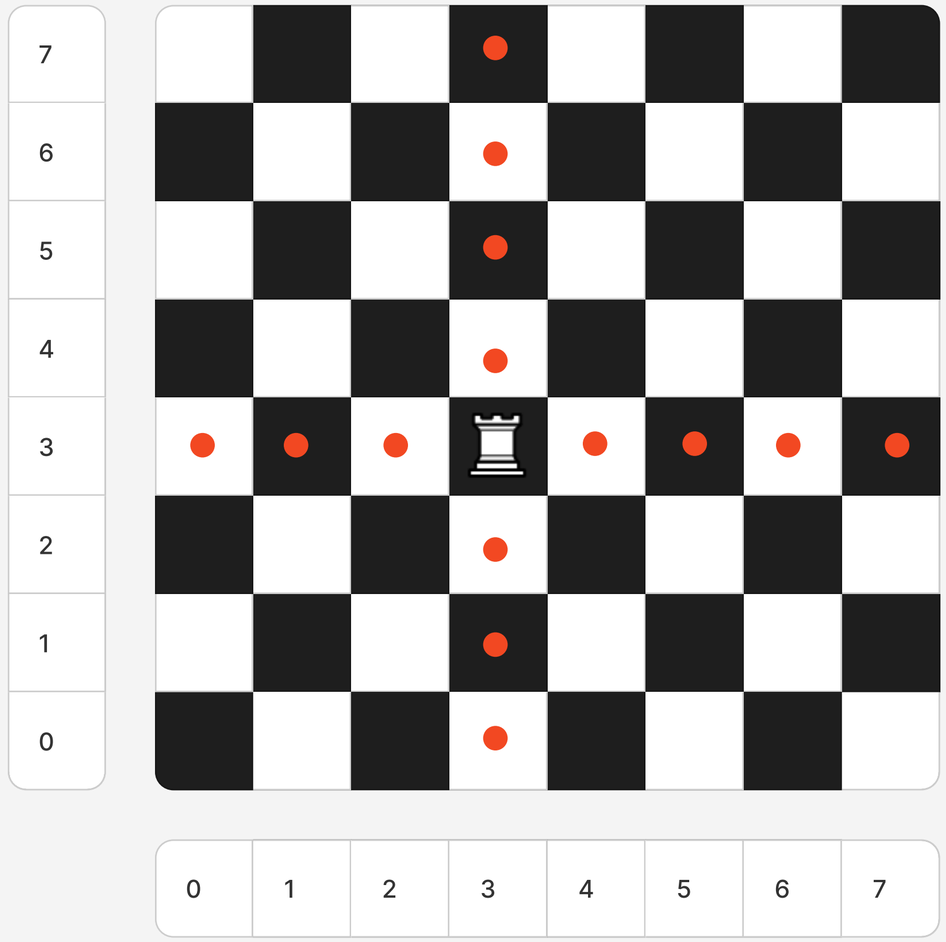
Bishops



Say we have a bishop in the center of the board on square (3,3), the possible squares a bishop is legally allowed to move to are listed below:
(2,2), (1,1), (0,0), (4,4), (5,5), (6,6) (7,7)

Difference between the current position (x1,y1) and the possible positions (x2,y2). (x1-x2)/(y1-y2)
(2,2), (1,1), (0,0), (4,4), (5,5), (6,6) (7,7) => 1,1,1,-1,-1,-1,-1
As seen above for a bishop to make a valid move, the diff is either 1 or -1.

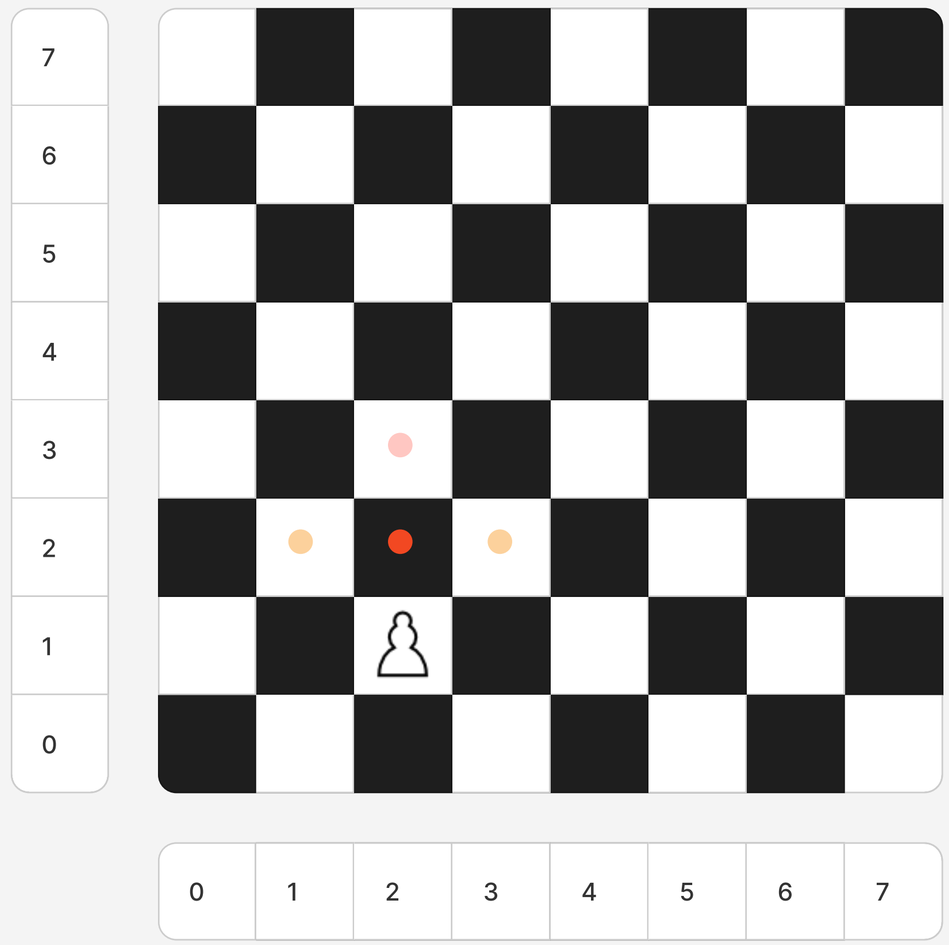
Rooks



Say we have a rook in the center of the board on square (3,3), the possible squares a rook is legally allowed to move to are listed below:
(4,3), (5,3), (6,3), (7,3), (2,3), (1,3), (0,3), and their compliments

Difference between the current position (x1,y1) and the possible positions (x2,y2). (x1-x2)/(y1-y2)
(4,3), (5,3), (6,3), (7,3), (2,3), (1,3), (0,3), and their compliments => This solution requires an iteration. Instead of taking the difference, one value is kept constant as the other is incremented or reduced with an upper limit of 7 and a lower limit of 0. To effectively check for all possible squares, a loop needs to traverse both the horizontal and vertical squares, keeping one constant and incrementing the other. In the case of validating a square, one diff needs to be 0 while the other is either negative or positive with an upper limit of 7 and a lower limit of 0. If both is 0 or none is 0, it is an illegal move.

Pawns



Pawns can move 2 squares if they are yet to move at all in the game, beyond that, they move only one step forward, and capture pieces one step diagonally. Pawns are tricky pieces to write a logic for owing to a rather ridiculous move called en passant. We will tackle the basic pawn moves here and deal with en passant much later.

Regular Pawn Move

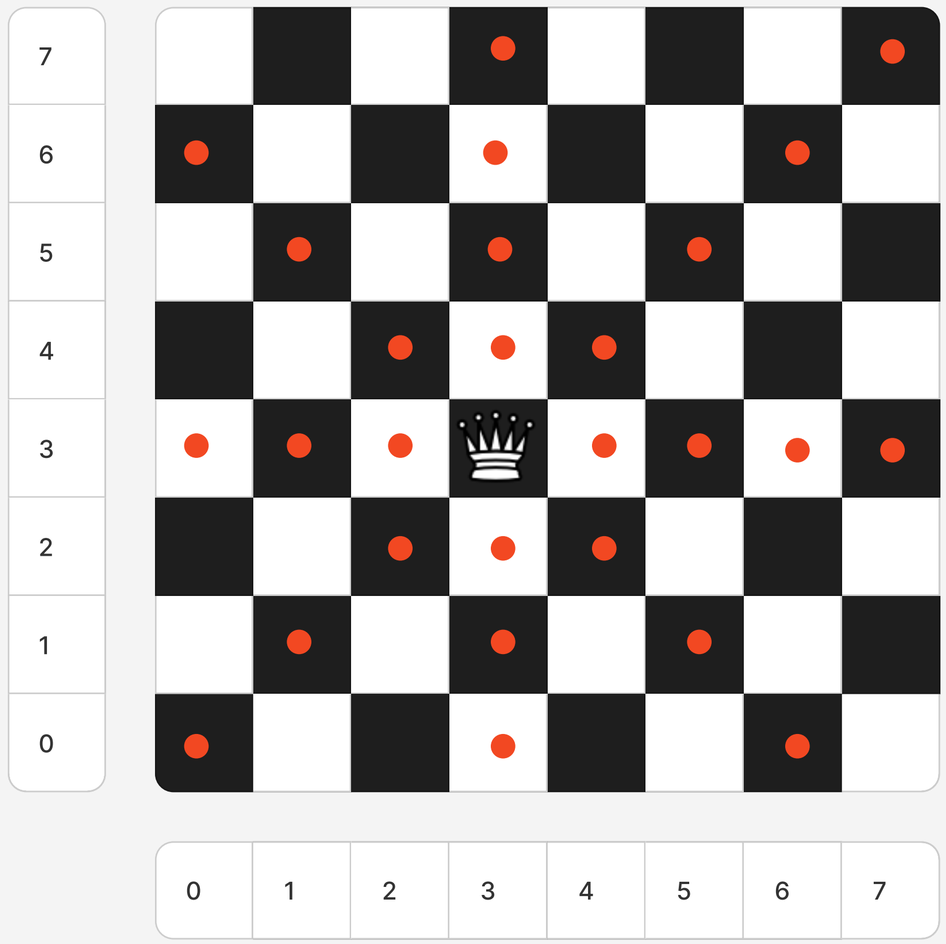
Assume the initial position of the pawn as (2,1). The possible positions are:

1. (2,2)
2. If the pawn has never moved before, (2,3) is a valid move.
3. If Opposition piece that is not a king is on either of the diagonals to the (2,1) square, (1,2) and (3,2) are valid moves.

Calculated Diff (x1-x2)/(y1-y2)

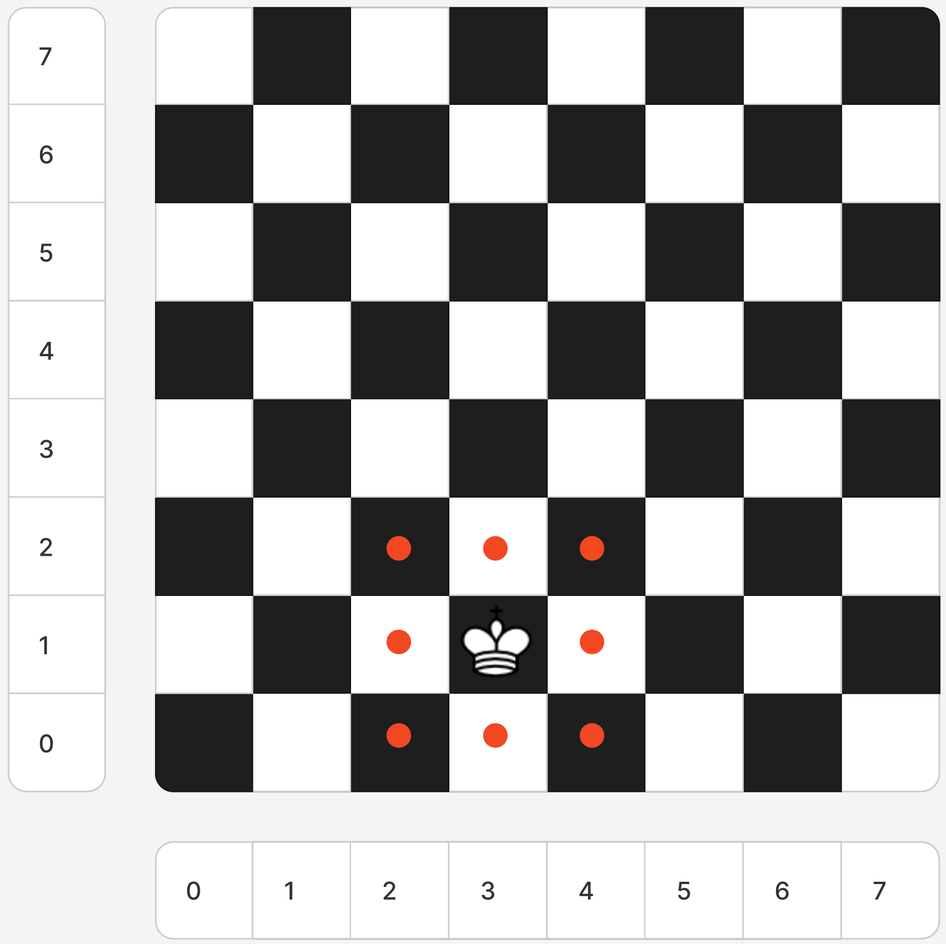
1. The difference in the x axis will be zero, and the y axis changes by 1 or -1 depending on the color of the piece
2. The difference in the x axis will be zero, and the y axis changes by 2 or -2 depending on the color of the piece.
3. On the diagonals, the calculated diff is either 1 or -1. Note that the sign will flip based on the color of the piece, since both white and black pawns traverse the board in opposite directions.

Queens



Queens are a combination of the Rook and the Bishop. The most powerful piece on the board, and also the easiest to write in code.

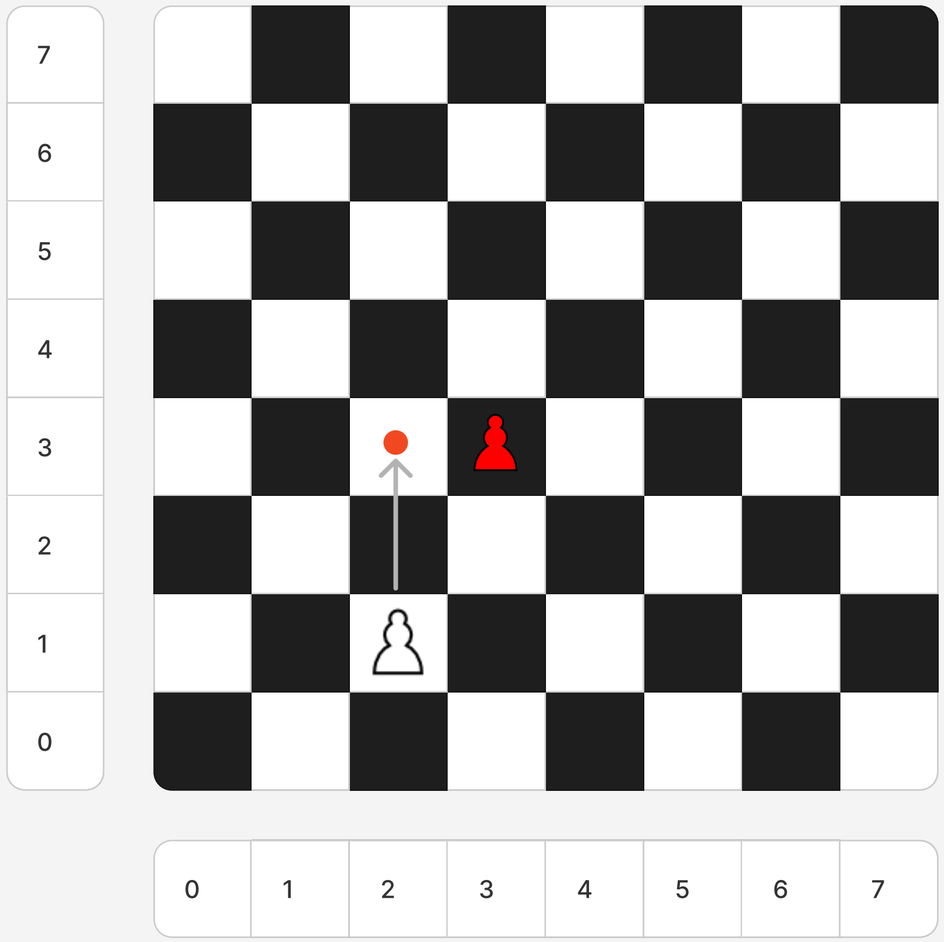
King

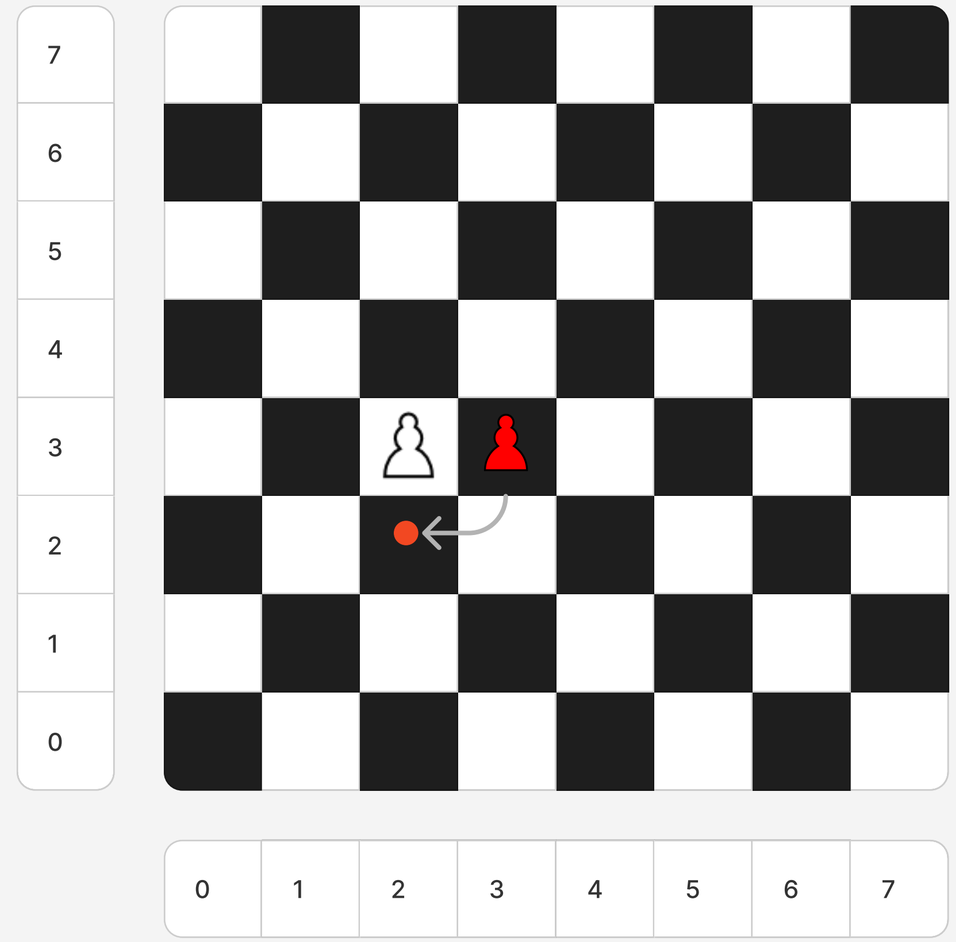


Arguably the laziest piece on the board, it however has complexities as it undergoes an interesting move called castling. This move is done in conjunction with a Rook if both of them are yet to move. Further details on this will be provided in the castling algorithm section.

The king mirrors a bishop and a rook as well, with the limitation of moving only 1 square at a time.

en passant



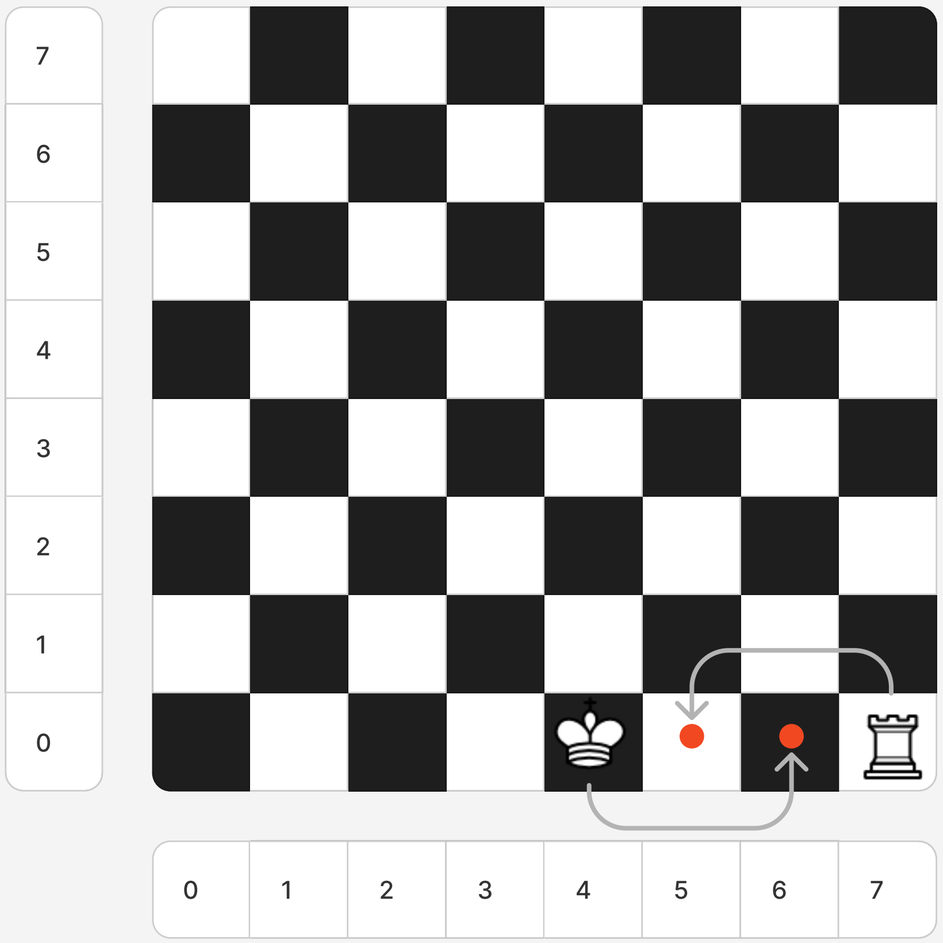


En Passant is tricky. We will examine it from the perspective of the opposition piece. En passant is a capture technique that is valid if the below conditions are met:

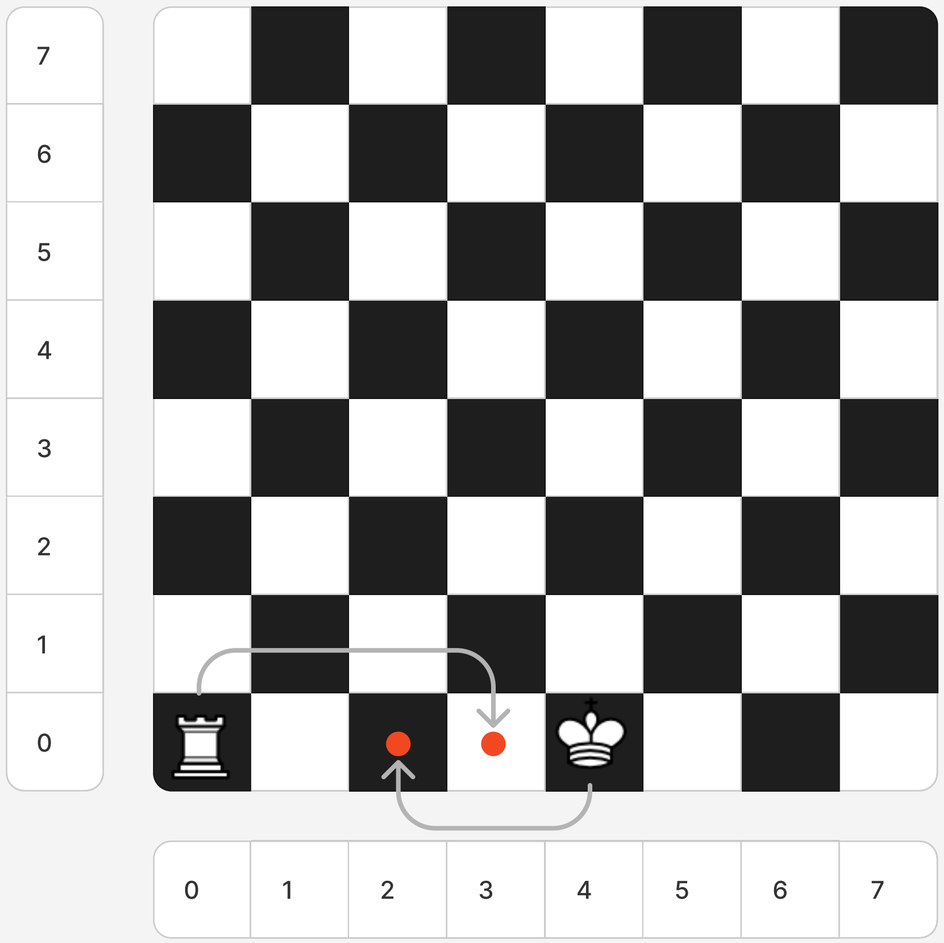
1. The pawn to be captured moved 2 squares from its inertia position.
2. The pawn to capture has crossed its own half of the board. In algebraic notation, white’s pawn should be on the 5th Rank. For Black, it will be the 4th Rank. In cartesian coordinate, it is Y = 4 for white and Y = 3 for black.

Given the above conditions, assume a black pawn(red pawn as seen above) is on the (3,3) square, a valid capture for black is the square (2,2). The diff is either 1 or -1 as expected for pawn captures. The critical conditions to consider are the two conditions above.

Castling



King side Castling



Queen side Castling

Castling is a defensive move to help protect the king in chess. For castling to be possible, the following conditions need to be met:

1. Both the King and the Rook to be castled haven’t made any initial move since the game started
2. There are no pieces between the king and rook.
3. The squares where either piece will land cannot be threatened
4. The King is not in check

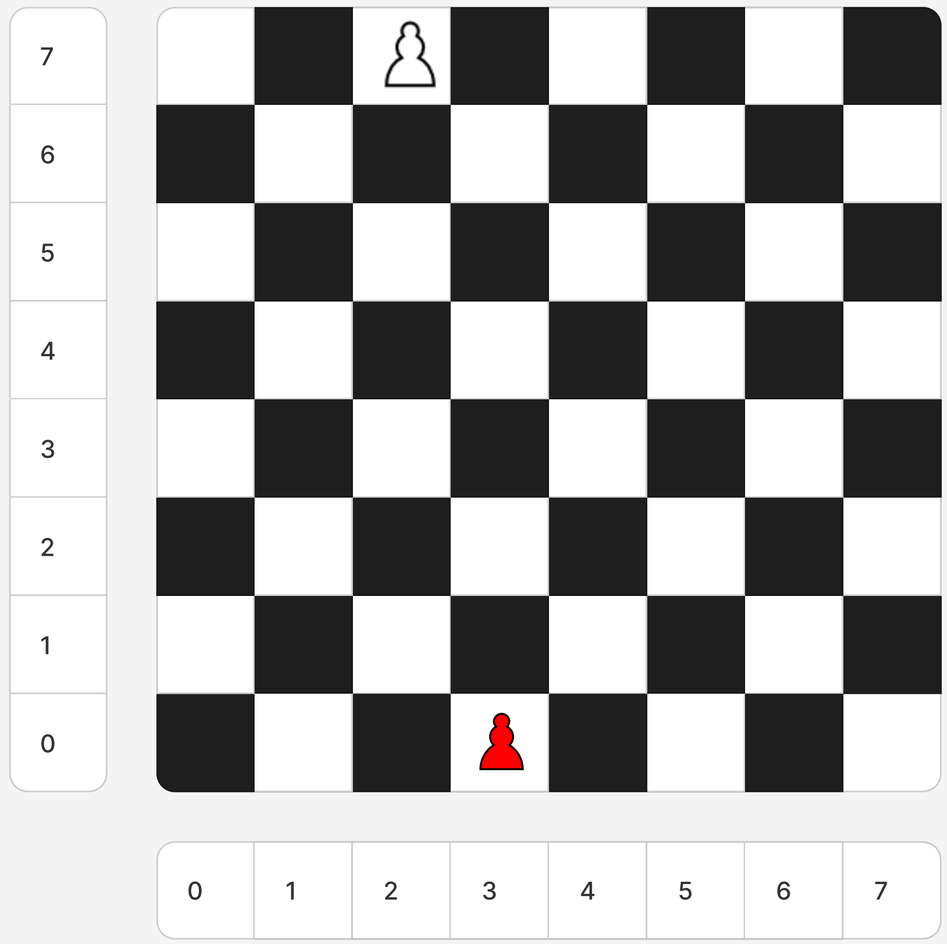
Once the above conditions are met, the easiest way to envision castling is to think of the king as moving 2 squares towards the rook, and placing the rook on the right hand(Queen side castling) or the left hand side(king side castling).

Initial position of king => (4,0). Proposed castling square => (6,0) {King side Castling}
Initial position of king => (4,0). Proposed castling square => (2,0) {Queen side Castling}
x-diff for king side castling => -2 for white pieces and 2 for black pieces
x-diff for king side castling=> 2 for white, and -2 for black
The y-diff for both castling remains 0 as the pieces don’t move vertically.

The rook’s new position for king side castling is always Initial position of king - (1,0) {White}
That of queen side castling is Initial position of the king + (1,0) {White}

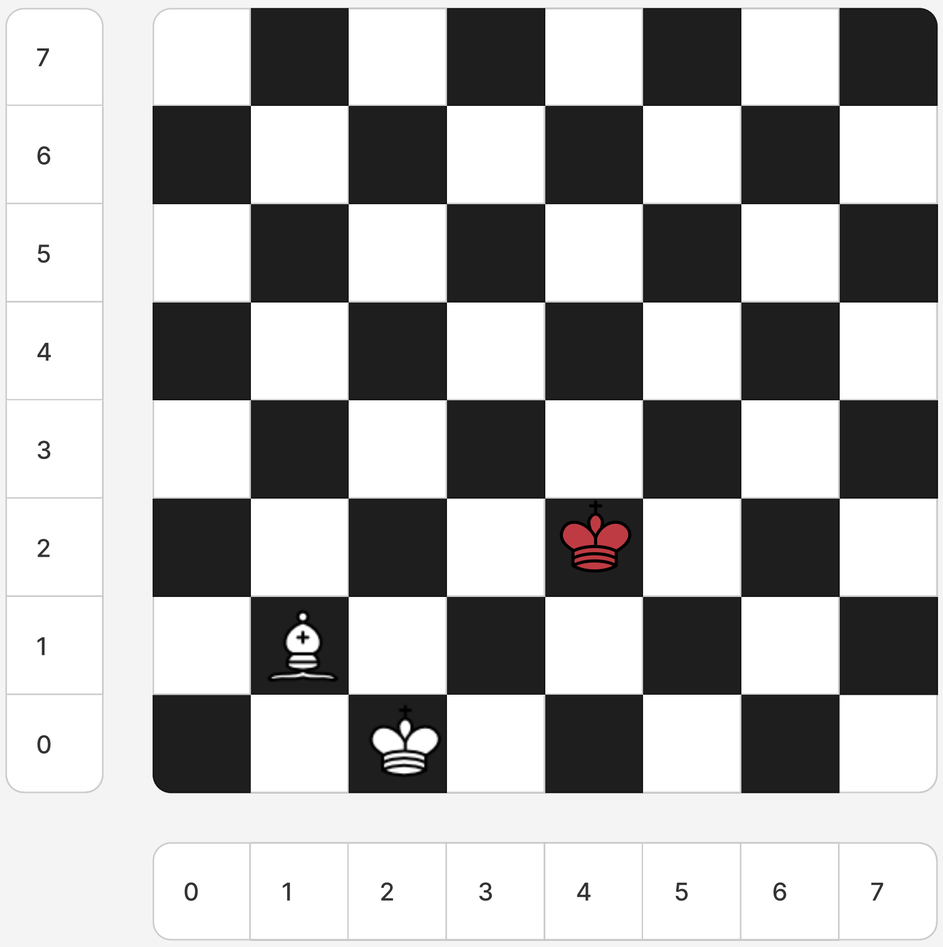
Remember to reverse the signs for black.

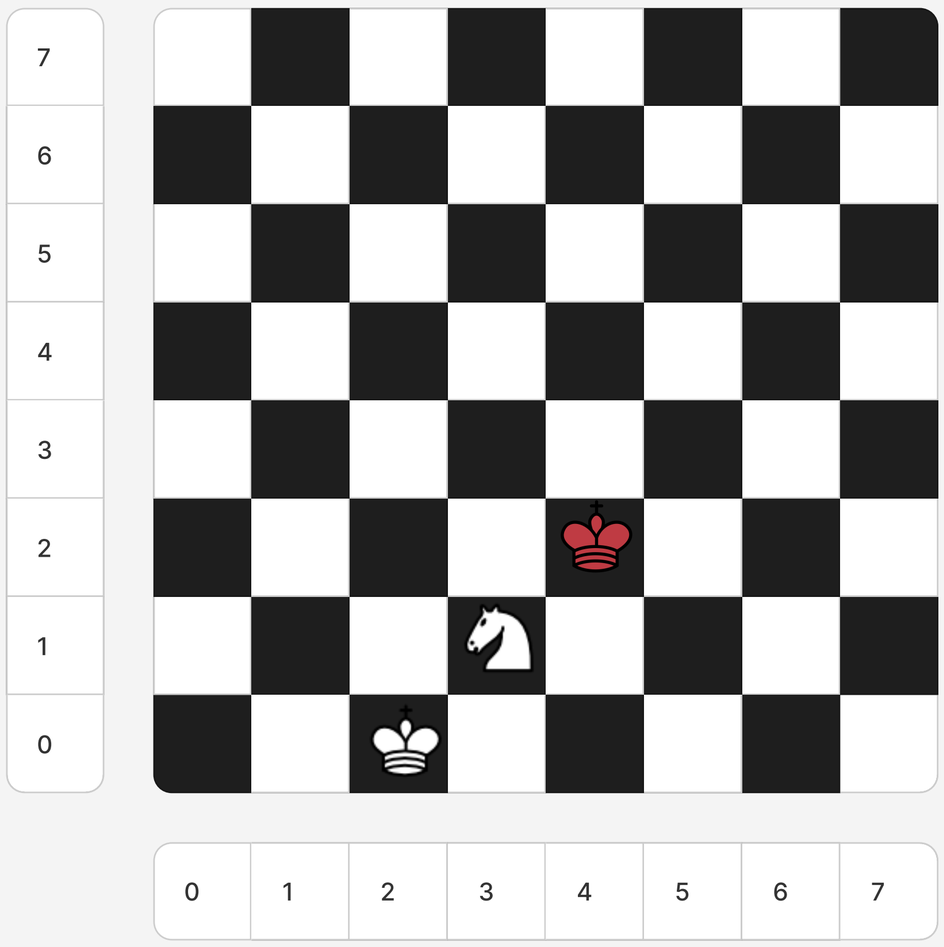
Pawn Promotion

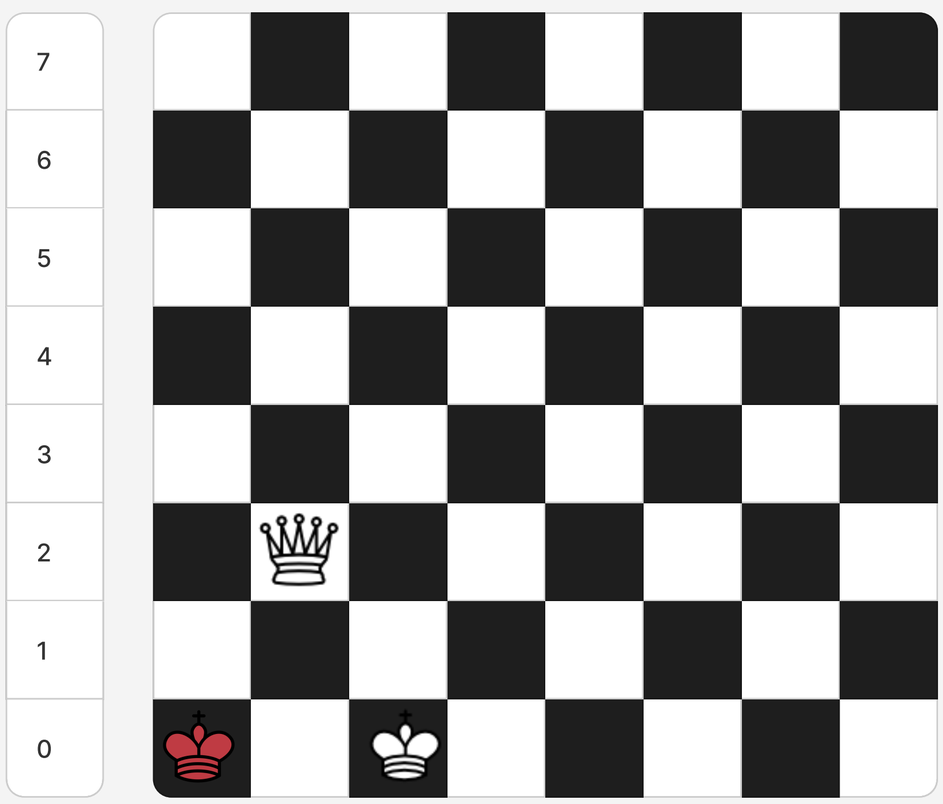


If a pawn gets to the very last square on the other side of the board, it becomes an officer. The player is allowed to select which of the officers they want their promoted pawn to be. The pawn can be anything but a king. There can be only one king on a chess board.
The only logic needed to validate this move is the Y coordinate. If a black pawn gets to Y => 0, it can be promoted. If a white pawn gets to Y => 7, it can be promoted.

StaleMate







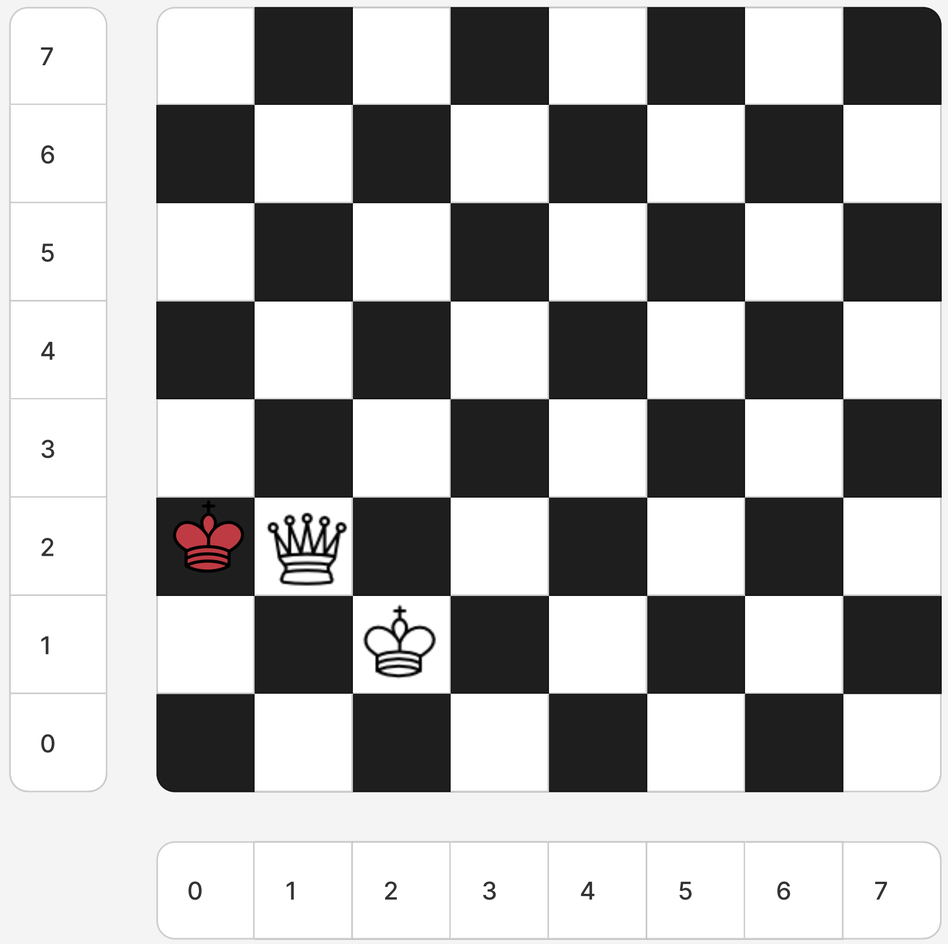
In this Case, Black's King(red) is to move but cannot move from the (0, 0) square without stepping into an attack. Hence, stalemate.

Williams Nimi Murray

Stalemate happens under a few conditions:

1. King is not under check but cannot move without entering a threatened square. (it has to be their turn to move)
2. When only a specific combination of pieces exist on the board
   a. **King and his Bishop VS the other king (Endless pursuits)**
   b. **King and his Knight VS the other King (Endless pursuits)**
   c. **Only both Kings (Bromance)**

Checkmate



Checkmate represents the end of the game. A game ends when the king is under check but is unable to move without entering a threatened square and has no piece available to defend the attack either by blocking the “threatpath” or killing the attacker

Logic

Pure Logic

Logic

Since this entire document aims to help anyone willing to learn a new language with a complete project to help learn critical parts of a language syntax, you can use google as your guide to write the barebones logic of this application. A few topics you will cover after completing the barebones logic in your desired language:

1. Variables
2. Functions
3. Conditionals
4. Loops
5. Basic Data Structures (needed to represent the board)
6. Basic mathematical calculations (Needed to calculate how the pieces move)
7. Trade off between space and runtime processing.
8. Exceptions
9. Importing packages
10. Utilizing Frameworks
11. Database
12. File I/O (Assuming you decide to save some logs and some data as json or other file formats)
13. API design
14. If you are feeling board enough, Frontend Development.

To make your work easier for you, you can translate each piece logic on paper before coding and using google as a guide.

Feeling Bold? Let’s expand this system. Next Page...

System Design and Architecture

If you are here, you have got that dawg in you!!!

This section is an attempt to build a large scale competitive chess platform using best practices. The Architecture and System design elements here are by no means exhaustive. As you implement, you can edit and add your relevant findings.

We shall explore this design in the following method.

1. Clearly State functional requirements
2. Clearly state the non-functional requirements
3. Create a sequence diagram showing relationship between the various entities and actors
4. Create a full system diagram showing what systems we have chosen to build this system with. Clear reasons for these selections will be given as side notes.

Walk with me...

Functional Requirements

1. A user must be able to create an account if they want without their email.
2. A user should be able to play as guest if they want.
3. A user should be able to login after creating their account.
4. The system should support single sign on (SSO)
5. All chess pieces should follow the standard chess logic rules that applies to them.
6. Users should be able to play a computer if they want.
7. Users should be able to see their ELO rating if they want
8. Users should be able to play with other users on the network if they want
9. Users should be able to play time controlled chess
10. Users should be able to see their game history and for each game history, see the moves history
11. Users be able to see their game play statistics
12. Users should be able to chat with other users during game play.
13. Users should be able to see when they joined the system
14. Users should be able to delete their account if they created one
15.

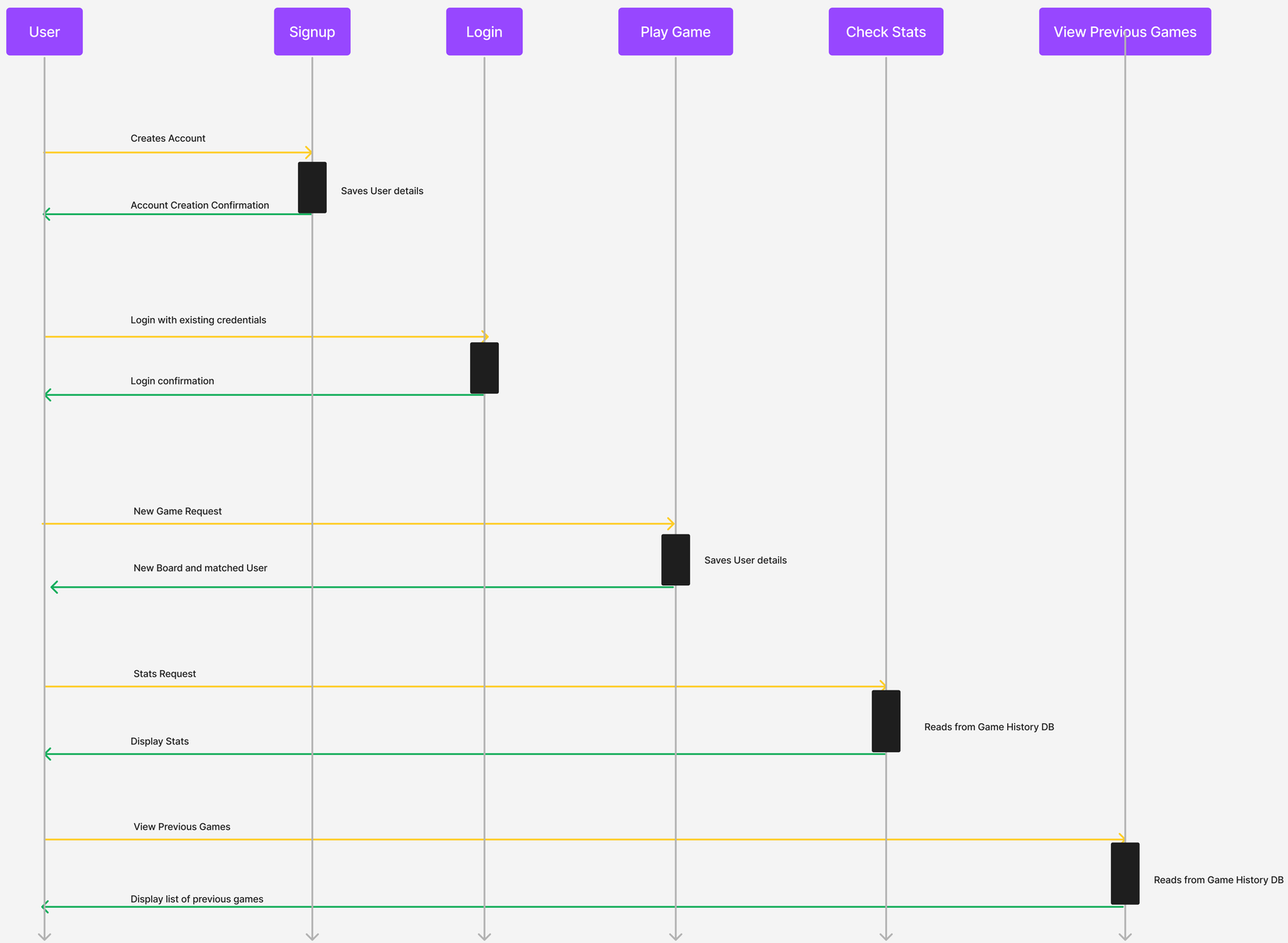
Walk with me...

Non-Functional Requirements

1. Scalability: The system should be able to scale to accommodate 10000 Users per second. The likes of Chess.com does about 16000 moves per second.
2. Performance: The system is expected to have low latency and high throughput. Latency of <30ms for login, and a game play latency of <30ms
3. Fault Tolerance: THe system should be able to handle failures and network partitions and users should be able to login and at least see their past games and also not lose games due to server failures
4. Availability: The system can manage 2 nines as it is not a mission critical system. That means there should be a downtime of about 4 days in a whole year.
5. Security: The system should be secure and should follow the OWASP top 10 security principles

Walk with me...

Sequence Diagram



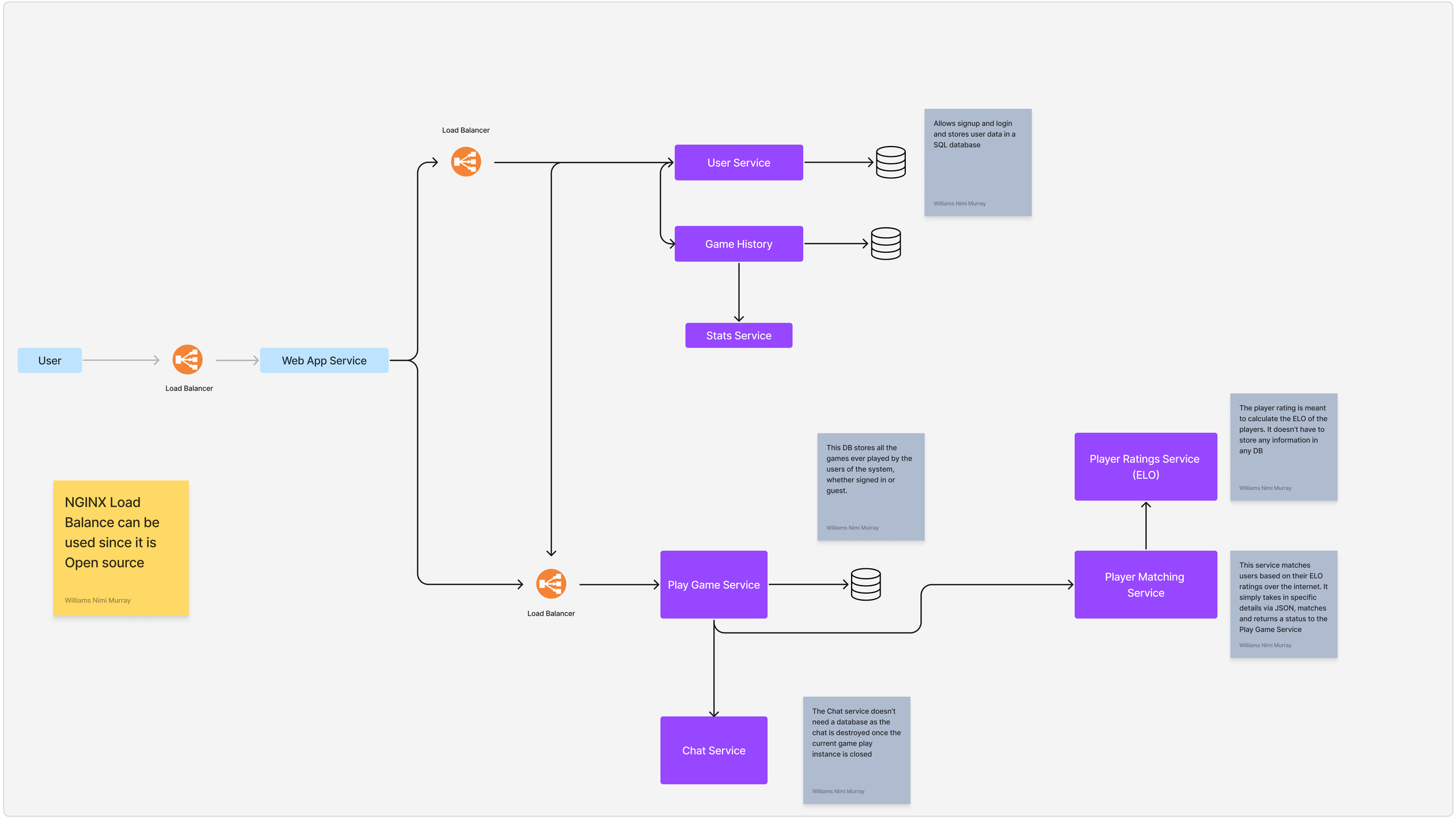
Preferred API Design architecture framework is GraphQL since the system will be making multiple calls to different services simultaneously

Alternative: REST API

API Request Methods:
Create Account: POST
Account creation Confirmation: PUT
Login: POST
Login Confirmation; PUSH
New Game Request: GET
New Board and Matched User: PUT
Stats Request: GET
Display Stats: PUT
View Previous Games: GET
Display List of Previous Games: PUT (API Pagination)

The Actual Game play will rely on WebHooks API given that the game play is in real time and information flow should be instant.

WebHook APIs will also be used to implement the chat service

Williams Nimi Murray

Architecture Diagram