

William Guerrand  
Online two-player chess, xiangqi, and shogi  
BSc. (Hons) Computer Science  
19<sup>th</sup> March 2021



## **Declaration**

I certify that the material contained in this dissertation is my own work and does not contain unreferenced or unacknowledged material. I also warrant that the above statement applies to the implementation of the project and all associated documentation. Regarding the electronically submitted version of this submitted work, I consent to this being stored electronically and copied for assessment purposes, including the School's use of plagiarism detection systems in order to check the integrity of assessed work. I agree to my dissertation being placed in the public domain, with my name explicitly included as the author of the work.

Date:

Signed:

## Abstract

There is a profuse amount of chess websites that exist for all kinds of variants. However, all website user interfaces are different which makes it quite tiresome and frustrating to move from one to the other. The purpose of this project is thus to solve this issue by creating Onechess, a chess website with a clean user interface that supports three of the most common variants: International chess, Xiangqi and Shogi. Since chess is prone to code re-use, this project also investigates how to actively benefit from using a modular approach to structure a program. By reading through this report, you will learn how to play all three variants and understand the process behind creating such a website from the design phase all the way down to the final solution.

## Table of Contents

Declaration .....	2
Abstract .....	3
1. Introduction .....	6
1.1 Motivation .....	6
1.2 Aims.....	6
1.3 Report Structure.....	6
2. Background .....	8
2.1 International Chess .....	8
2.1.1 Board & Pieces .....	8
2.1.2 Piece Movement .....	9
2.1.3 Captures.....	10
2.1.4 Special moves.....	11
2.1.5 Promotion .....	13
2.1.6 End of game .....	13
2.2 Xiangqi .....	14
2.2.1 Board & Pieces .....	14
2.2.2 Piece Movement .....	15
2.2.3 Captures.....	17
2.2.4 End of game .....	18
2.3 Shogi .....	19
2.3.1 Board & Pieces .....	19
2.3.2 Piece Movement .....	20
2.3.3 Captures.....	22
2.3.4 Special Rules .....	22
2.3.5 End game.....	22
2.4 Analysis of similarities between variants .....	23
2.5 Analysis of existing websites.....	24
3. System Design.....	28
3.1 Section aims.....	28
3.2 System entities & Attributes.....	28

---

3.3	Database Design .....	29
3.4	Flow of Execution .....	30
3.5	Game Data Architecture .....	32
3.6	User Interface .....	34
3.7	Requirements .....	36
4.	Implementation.....	38
4.1	Section Aims.....	38
4.2	Development Methodology .....	38
4.3	Key Implementation Stages .....	39
4.3.1	Basic Server-Client .....	39
4.3.2	Server-Client Communication.....	40
4.4	System in Operation .....	41
5	CONCLUSION .....	45
	Acknowledgments .....	47
	Bibliography .....	48

# 1. Introduction

## 1.1 Motivation

This project has come about because of two issues that appear when searching online for a chess website. The first is that websites all have their own user interfaces and mechanics. After one gets comfortable with the feel of a website it can be quite frustrating to learn everything anew when shifting to another one. The second is that, at the present time, a chess website will most likely specialize in one chess variant. Therefore, if a user wants to play shogi and xiangqi on one website, then that user has no choice but to create one account for each website. Not only is doing the same task twice futile but it will also take more effort as the user will have to go through the learning process of understanding how to use a website once more which is irritating and a waste of time. This demonstrates the necessity to have a website that combines multiple chess variants.

Another reason that explains why this project is worth looking at and might be helpful to others is code re-use and how effective it can be in certain scenarios. In our case, it is very much pertinent because of the very nature of the subject matter. Section 2.4 will take a closer look at why this project is so easily prone to code-reuse and how it plays an important role.

## 1.2 Aims

Creating a website where a user can play xiangqi, international chess and shogi is the most important aim of this project. Another is to make proper usage of code inheritance, modularity and code re-use concerning the three variants. The third is to have a user interface that is simple to operate, making it easy to jump from one variant to the other. The hope is - if this were to be put into production - that users would find it simpler to play other versions of chess that are different from the ones they are accustomed to because of how accessible it has become, expanding their horizons and perhaps even learning something new about someone else's culture.

## 1.3 Report Structure

This report will be split into several sections. Each section builds on top of the previous one and will frequently follow chronological order. By doing this I hope to be able to give insight into the thought process behind choices that resulted in the solution that I will be presenting at the end. 'Background' is the first section, and its purpose is to help the reader get a better grasp of what this project is about through three main subsections. The first is a tutorial to comprehend the rules behind all three variants. The second is an analysis of chess websites that already exist and what dos and don'ts we can draw from them. The third is an assessment of how these three variants are good candidates for code re-use. 'System design' is the second section and it is the foundation for implementation. Its purpose is to investigate and conceptualize at a high level what tools and entities are necessary to accomplish our aims. 'Implementation' goes in depth about the actual act of development. It takes the entities and vague ideas from the 'System design' and puts them into

practice. The 'Conclusion' ties everything together by pointing back to our original aims section and evaluate whether the aims were achieved or not.

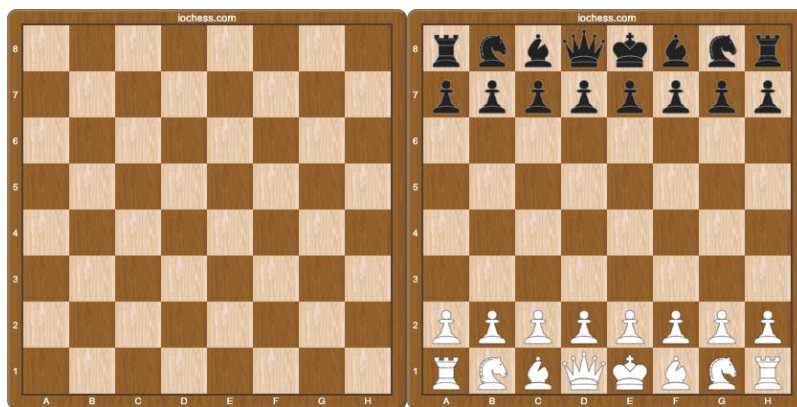
## 2. Background

Before we move on, it is important to define a few terms. A chess variant, or variant for short, can be defined as a variation of international chess where the goal remains to capture an opponent's leader piece. Shogi and Xiangqi are variants. The game of chess can be defined as a strategy board game where two players take turns to move pieces on a board to attack the opponent's leader piece so that it cannot make a legal move, this is checkmate: the leader piece is captured which means the player responsible for that piece has failed to protect it and thus loses. Time control can be added to a game of chess. This forces the players to make a move, accelerating the pace of the game. The player whose time ran out loses the game. Increments can also be added at each turn to add more time to one's clock.

This section has three aims. The first is to explain the rules of xiangqi, shogi and international chess. The second is to understand why these three variants are great for code re-use. The third is to discuss any relevant websites that already exist and what dos and don'ts we can draw from them.

### 2.1 International Chess

#### 2.1.1 Board & Pieces



*Figure 2.1.1.1 – left: empty International Chess board, right: set International Chess board*



*Figure 2.1.1.2 – International Chess pieces; from left to right: king, queen, rook, bishop, knight, and pawn*

The board, as shown in Figure 2.1.1.1, has 8 horizontal squares and 8 vertical squares. The squares on top of the board form a checkered pattern. The checkered pattern has no impact on the



way pieces move, they are decorative and helpful to clearly outline squares. There are 6 different pieces as shown in Figure 2.1.1.2. When pieces are set one player has 8 pawns, 2 rooks, 2 knights, 2 bishops, 1 queen and 1 king. Each piece type has certain rules attached to it. Let us see what they are.

### 2.1.2 Piece Movement

For illustration purposes, I will display the piece's legal moves and captures by using a black pawn.

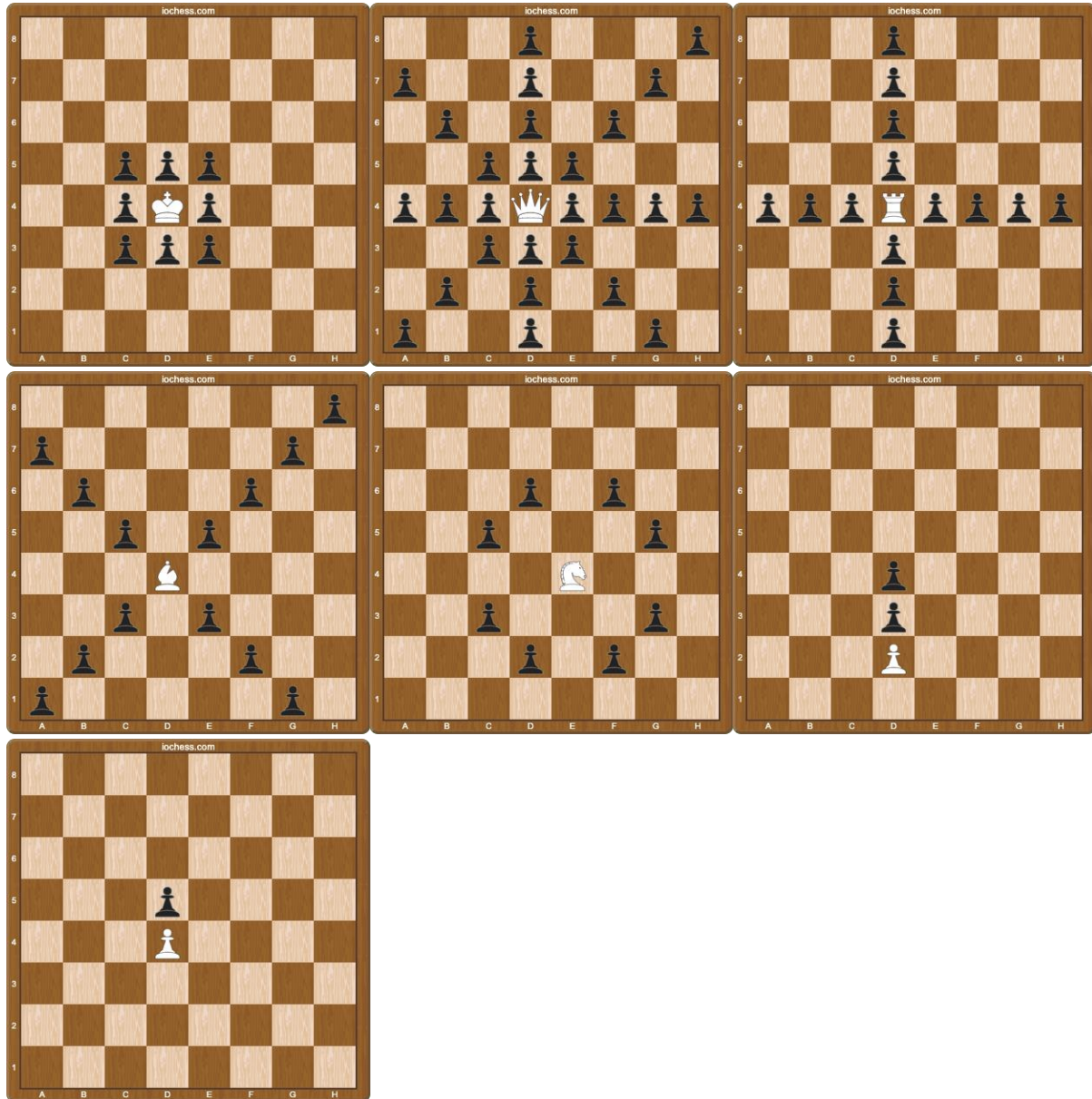


Figure 2.1.2.1 – top-left: king's valid moves, top-middle: queen's valid moves, top-right: rook's valid moves, middle-left: bishop's valid moves, middle-center: knight's valid moves, middle-

*right: pawn's valid moves on starting square, bottom-left: pawn's valid moves after leaving starting square*

Each square can only be occupied by one piece. A piece cannot capture another piece if it is of the same color. A piece cannot be placed outside of the board. A piece cannot jump over another piece except for the knight as shown in Figure 2.1.2.1. If a piece lands on a square occupied by the opponent's piece it can capture it by moving it to the side. Once removed the piece is no longer able to come back on the board.

A **King** moves one square in any direction, so long as that square is not attacked by an enemy piece, see Figure 2.1.2.1. Kings can also do castling (see 2.1.4).

A **Queen** moves diagonally, horizontally, or vertically any number of squares, see Figure 2.1.2.1.

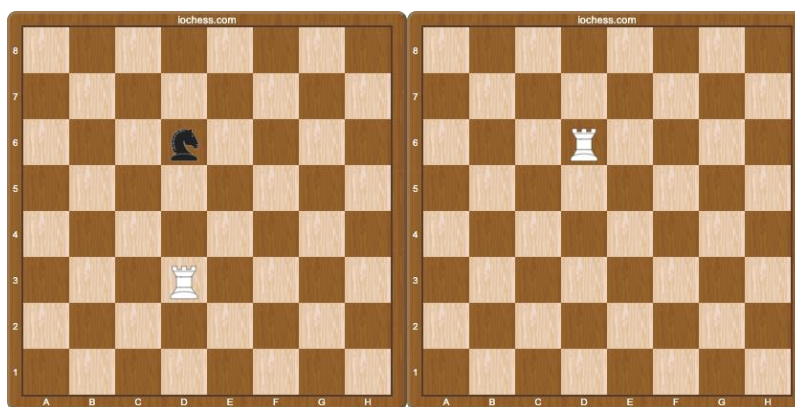
A **Rook** moves any number of squares orthogonally, see Figure 2.1.2.1. Rooks participate in castling (see 2.1.4).

A **Bishop** moves diagonally any number of squares until it hits the edge of the board or a piece, see Figure 2.1.2.1.

A **Knight** moves in an 'L' shape: two squares in a horizontal or vertical direction, then move one square horizontally or vertically, see Figure 2.1.2.1. It is the only piece that can jump over another piece and works differently than in xiangqi's horse (see Figure 2.2.2.1).

A **Pawn** moves vertically forward one square, with the option to move two squares if they have not yet moved, see Figure 2.1.2.1. Pawns capture differently to how they move. They capture one square diagonally in a forward direction. Pawns must promote upon reaching the other side of the board and can morph into any other piece, except for a king. Pawns are involved in "En-Passant" (see 2.1.4).

### 2.1.3 Captures



*Figure 2.1.3.1 – rook captures knight in International Chess*

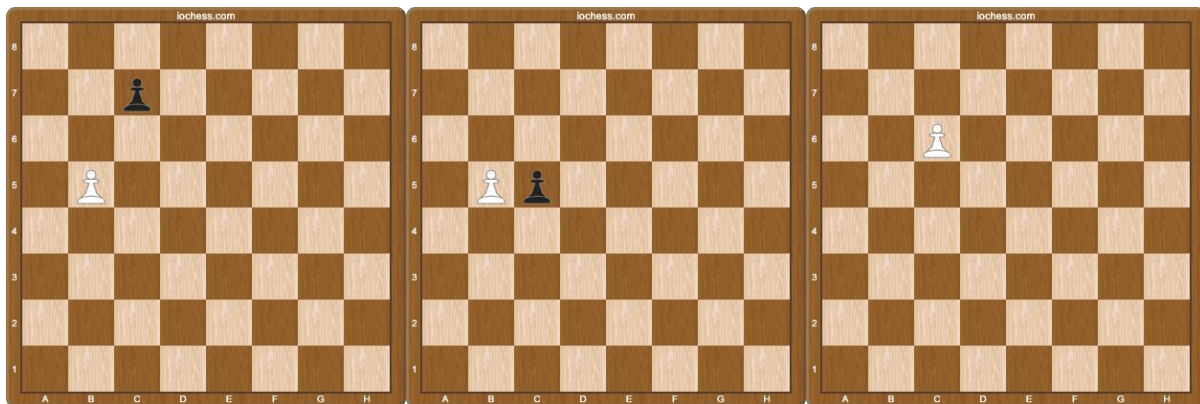
Most pieces capture in the direction of movement, see Figure 2.1.3.1, pawns being the exception.



*Figure 2.1.3.2 – pawn captures rook in International Chess*

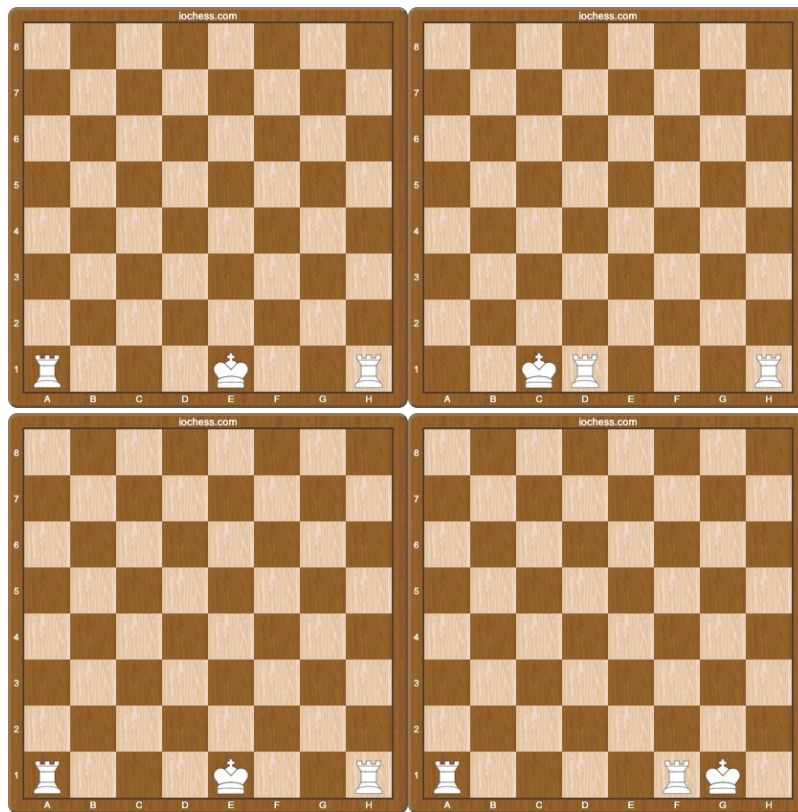
Looking at Figure 2.1.3.2, the first state shows a white pawn which cannot move any further as it is blocked by an enemy piece. This is different from shogi and xiangqi pawns (see Figure 2.2.2.1 and Figure 2.3.2.1). In the second and third game state we show how the pawn captures the rook sideways.

#### 2.1.4 Special moves



*Figure 2.1.4.1 – a pawn capturing in “en-passant” in International Chess*

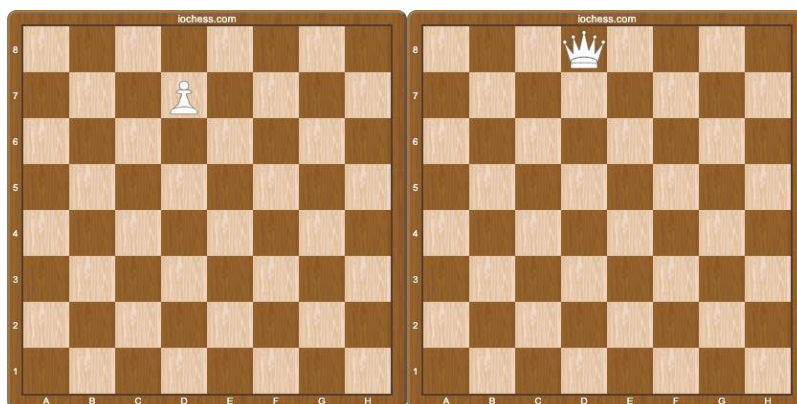
**En-passant** is when a pawn captures another pawn in a special way. The condition for this capture is that the opponent's last move is a two-step forward pawn move, see Figure 2.1.4.1.



*Figure 2.1.4.2 –top: a king long-castling, bottom: a king short-castling*

**Castling** is a special move where the king hides behind a rook which looks like a fortress, see Figure 2.1.4.2. It can only be done if the pieces between the king and the rook are vacant. Moreover, it can only be done if the king and the rook has not moved. Since there are two rooks there are two possible ways of castling: short-castle and long-castle. In either case the king moves two squares in the direction of the rook while the rook simply moves to the square next to the king. It is important to note that the squares that the king lands on or traverses to get to that square cannot be attacked by another piece.

### 2.1.5 Promotion



*Figure 2.1.5.1 – a pawn promoting to a queen in International Chess*

**Promotion** in International chess is obligatory unlike shogi (see 2.3.4). It occurs after a pawn reaches the far end of the board, see Figure 2.1.5.1. The player can then choose into what piece the pawn will turn into, rook, knight, bishop, or queen. Pawns will most often be promoted to queens as it is the most powerful piece type in International Chess.

### 2.1.6 End of game

A game can end in either a checkmate, stalemate, or a draw. Stalemate is a board state where the player is unable to make a valid move and where the king is not attacked, it is a draw. Both players can also agree on a draw if they want to. Other rules exist that force the game into a draw, notably the 50-move rule and the three-fold repetition.

## 2.2 Xiangqi

### 2.2.1 Board & Pieces

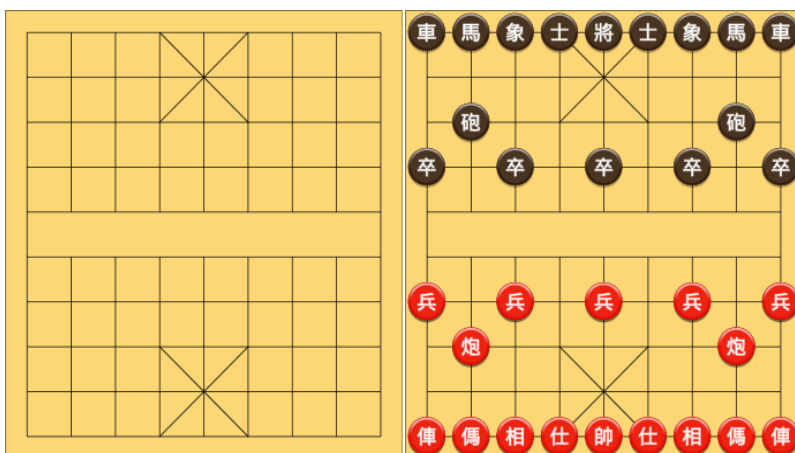
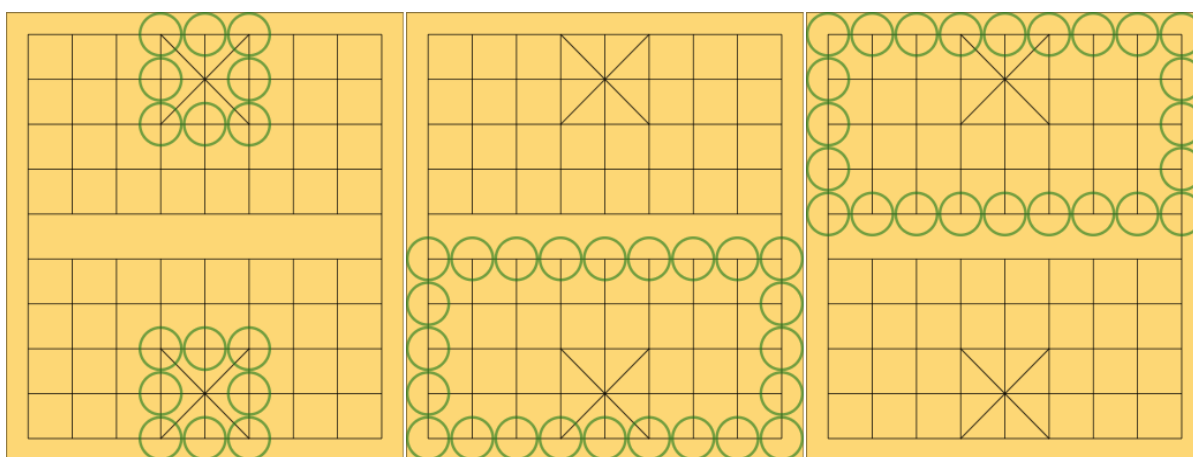


Figure 2.2.1.1 – left: empty Xiangqi board, right: set Xiangqi board



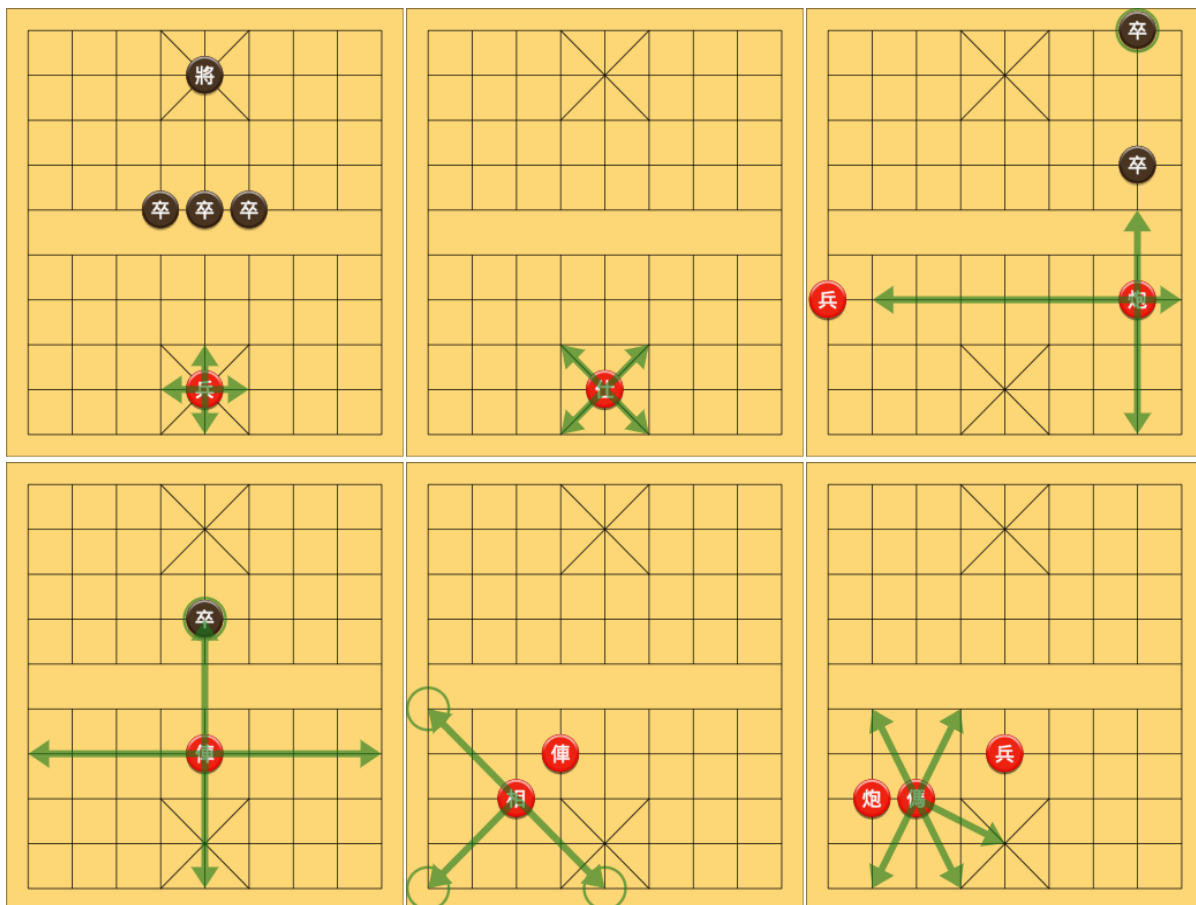
Figure 2.2.1.2 – Xiangqi pieces; from left to right: general, advisor, catapult, rook, horse, elephant, and pawn



2.2.1.3 – Xiangqi board zones delimited by circles; left: palace, center: land before river, right: land after river

The board, as shown in Figure 2.2.1.1, has 9 horizontal points and 10 vertical points. Unlike International Chess, we do not say squares but rather points since the pieces sit on top of square intersections. Moreover, the board has three zones, and these have an impact on the way the game is played, see section 2.2.1.3 to see the zones and 2.2.2 to understand how the zones matter when considering piece moves. The line of emptiness in the middle of the board is called the river. The small square of 3x3 on either side of the board is called the palace. The land on either side of the river is land that belongs to either red or black depending on the player perspective. There are 7 different pieces as shown in Figure 2.2.1.2. When pieces are set one player has 5 pawns, 2 rooks, 2 horses, 2 elephants, 2 catapults, 2 advisors and 1 general. Each piece type has certain rules attached to it. Let us see what they are.

### 2.2.2 Piece Movement



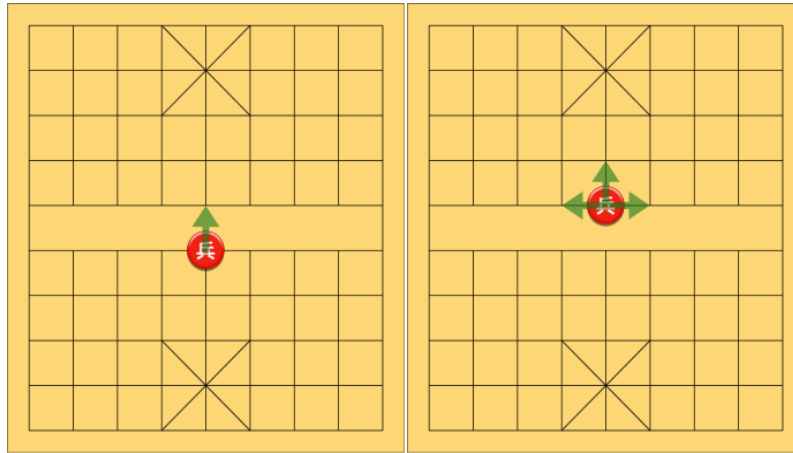


Figure 2.2.2.1 – top-left: king's valid moves, top-middle: advisor's valid moves, top-right: catapult's valid moves, middle-left: rook's valid moves, middle-center: elephant's valid moves, middle-right: horse's valid moves, bottom-left: pawn's valid moves before river, bottom-right: pawn's moves after river

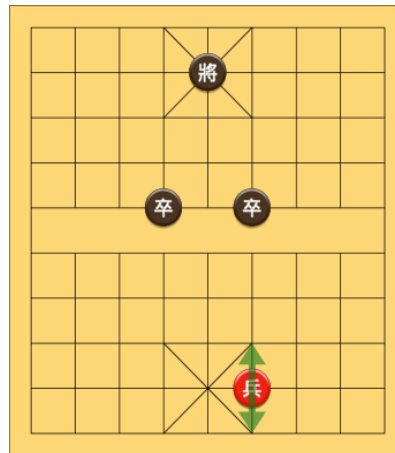


Figure 2.2.2.2 – king valid moves, arrangement showing opposing general's rule being applied

Each point can only be occupied by one piece. A piece cannot capture another piece if it is of the same color. A piece cannot be placed outside of the board. A piece cannot jump over another piece except for the catapult as shown in Figure 2.2.2.1. If a piece lands on a point occupied by the opponent's piece it can capture it by moving it to the side. Once removed the piece is no longer able to come back on the board.

A **General** moves one point orthogonally, see Figure 2.2.2.1, so long as: it is within the palace, the landing point is not attacked by an enemy piece, it does not lead to the generals facing each other as seen in Figure 2.2.2.2.

An **Advisor** moves diagonally so long as it is within the palace, see Figure 2.2.2.1.

A **Rook** moves any number of points orthogonally, see Figure 2.2.2.1. It is considered the most powerful piece on the board.



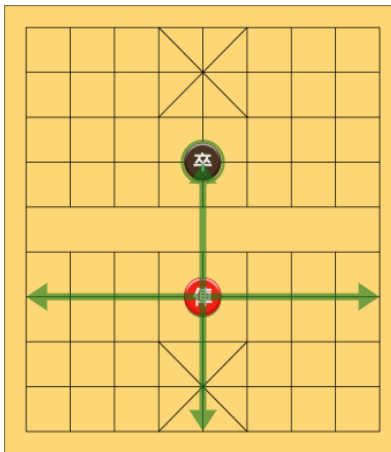
A **Catapult** moves any number of points orthogonally, see Figure 2.2.2.1. It is different to the xiangqi rook as it does not capture in the direction of the movement. A catapult can only capture by jumping a single piece of either color along the path of attack. Any number of unoccupied points may exist between the piece being attacked and the piece being jumped over.

An **Elephant** moves exactly two points diagonally, see Figure 2.2.2.1, as long as: there is no piece in front of the direction of movement, it stays in friendly territory (land before the river).

A **Horse** moves exactly like a knight in International Chess (see Figure 2.1.2.1), one point orthogonally followed by one point diagonally away from its original position, see Figure 2.2.2.1. The difference is that it cannot jump over a piece that sits in the path of movement.

A **Pawn** moves forward by one before the river. It captures in the direction of the movement, unlike International Chess. After crossing the river, the pawn can move and capture on both of its sides.

### 2.2.3 Captures



*Figure 2.2.3.1 - rook captures pawn in Xiangqi*

Most pieces capture in the direction of movement, see Figure 2.2.3.1, catapults being the exception, see Figure 2.2.2.1.

#### **2.2.4 End of game**

A game can end in either a checkmate, forced win or a draw. Forced draw is the equivalent of stalemate in International Chess (see section 2.1.6) except that the player who is unable to move loses. Both players can also agree on a draw if they want to.

## 2.3 Shogi

### 2.3.1 Board & Pieces

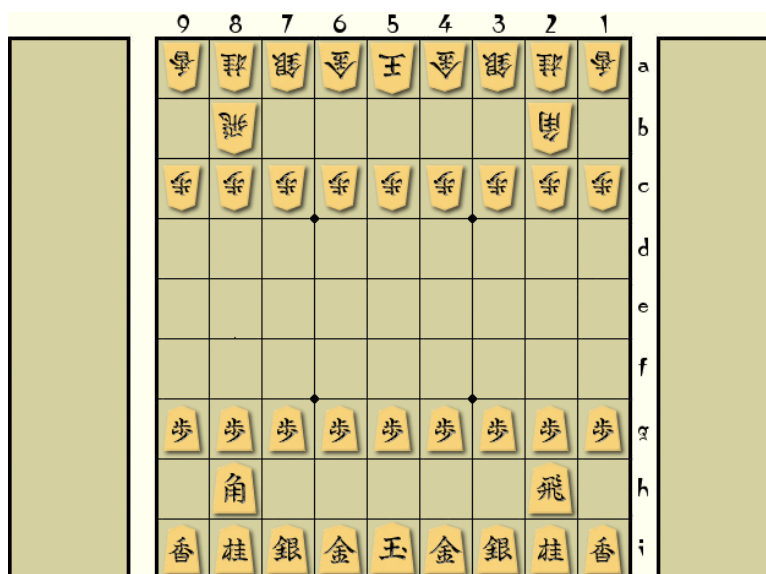


Figure 2.3.1.1 - set Shogi board

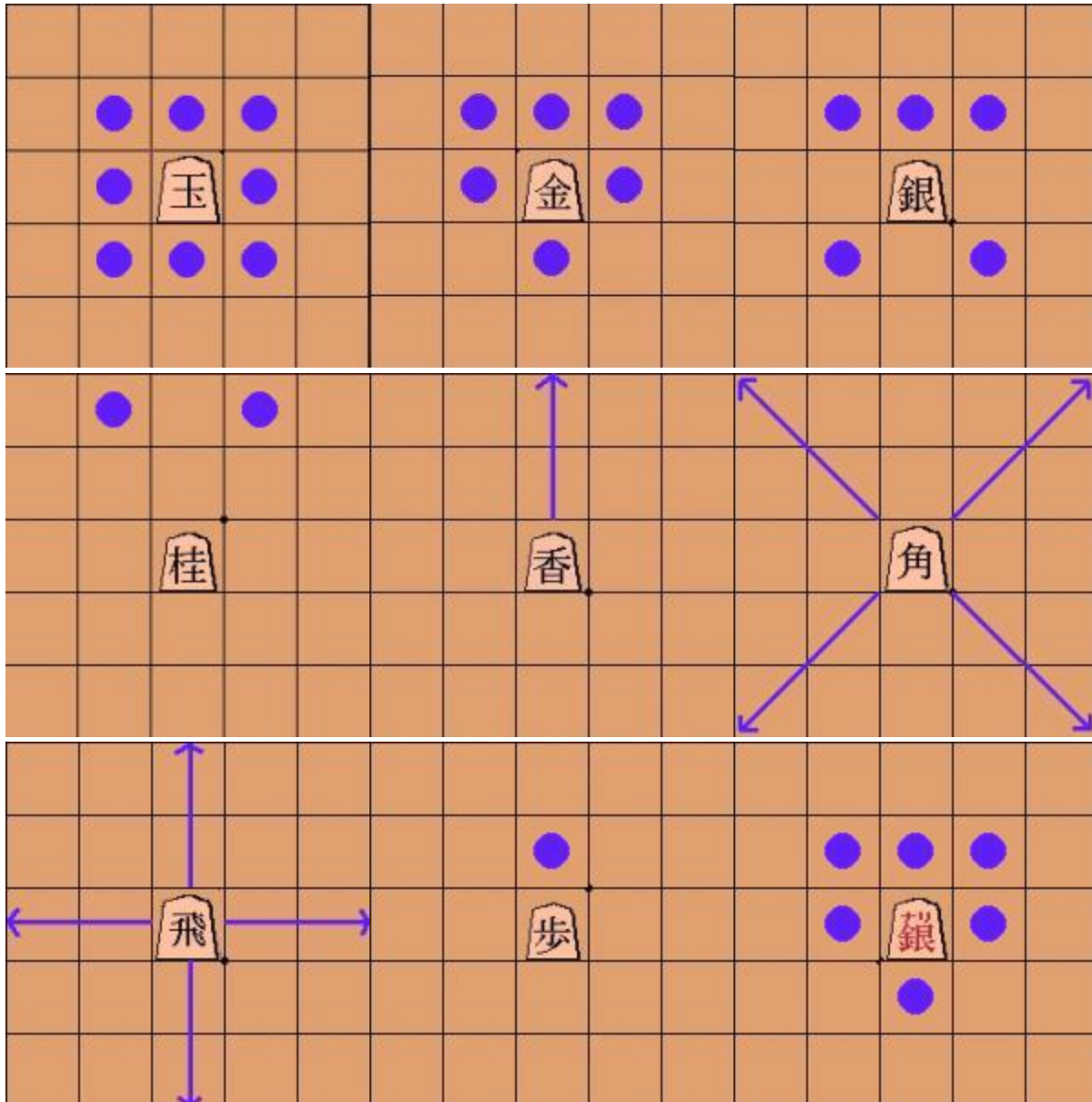


2.3.1.2 - Shogi pieces; all top pieces are unpromoted: from left to right: king, gold, silver, horse, lance, bishop, rook and pawn; all bottom pieces are promoted: for left to right: silver, horse, lance, bishop, rook and pawn

The board, as shown in Figure 2.3.1.1, has 9 horizontal squares and 9 vertical squares. On both sides of the board are sideboards. This is where captured pieces are kept. Pieces that are kept on the sideboard can be dropped on the board which counts the same as a move. This is different from International Chess and adds another layer of complexity to the game. Since capturing and placing pieces back on the board is possible this necessitates that both opponent pieces and friendly pieces are identical. To recognize which pieces belong to which player the orientation of the piece provides a hint. A way to remember it is your pieces will be pointing towards your opponent and his pieces will be pointing towards you. The promotion zone spans three rows behind the two dots on the opposite side of the board, the details of how promotion works is in section 2.3.4. There are 8 different pieces as shown in Figure 2.3.1.2. When pieces are set one player has 9 pawns, 2 lances, 2 horses, 2 gold, 2 silver 1 bishop, 1 rook and 1 king. The design on

the king piece can change and this indicates which player is the challenger and which player is more proficient. Each piece type has certain rules attached to it. Let us see what they are.

### 2.3.2 Piece Movement



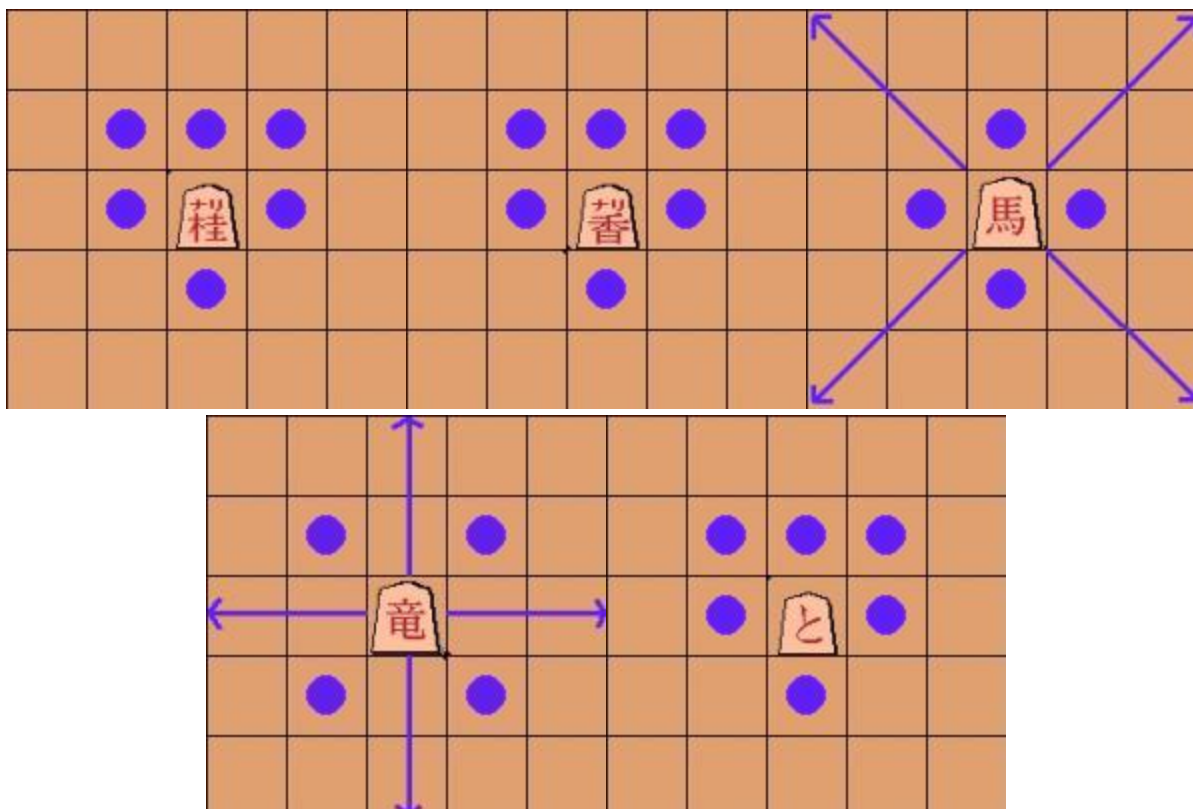


Figure 2.3.2.1 – top-left to bottom right: king's valid moves, gold's valid moves, silver's valid moves, horse's valid moves, lance's valid moves, bishop's valid moves, rook's valid moves, pawn's valid moves, promoted silver's valid moves, promoted horse's valid moves, promoted lance's valid moves, promoted bishop's valid moves, promoted rook's valid moves, promoted pawn's valid moves

A **King** moves one square in any direction so long as the landing square is not attacked by an enemy piece.

A **Gold** moves one square in any direction apart from both back-diagonal squares.

A **Silver** moves one square in any direction except sideways and backwards.

A **Horse** moves two squares ahead and then one sideways

A **Lance** moves any number of squares forward.

A **Bishop** moves any number of squares diagonally.

A **Rook** moves any number of squares orthogonally.

A **Pawn** moves one square forward.

A **Promoted Silver** moves like a gold.

A **Promoted Horse** moves like a gold.

A **Promoted Lance** moves like a gold.

A **Promoted Bishop** moves like a bishop and like a king.

A **Promoted Rook** moves like a rook and like a king.

A **Promoted Pawn** moves like a gold.

### 2.3.3 Captures

All pieces capture in the direction of movement.

### 2.3.4 Special Rules

Promotion works in a similar fashion than International Chess in the sense that a piece can be promoted at the end of one's turn. However, the differences are numerous, and they go as follows. To reiterate, the promotion zone is the combination of the three furthest rows from the player. It is only possible to promote a piece if it is in the promotion zone or leaving it. A piece isn't forced to promote unless it doesn't have any valid moves after being placed on the destination square regardless of the pieces surrounding it. If a piece is promoted it remains that way until it is captured or until the game ends. Gold and king are the only pieces which cannot be promoted.

The sideboard is used to place captured pieces. When it is a player's turn to play, instead of making a move on the board a player can move a piece from the sideboard to the board, this is called dropping. A piece can only be dropped onto a vacant square with its unpromoted face up. Pieces cannot be promoted when dropped. A pawn cannot be dropped onto a column which already contains a pawn belonging to the same player. A pawn cannot be dropped to deliver checkmate. A piece may not be dropped on a square from which it will have no valid moves regardless of which pieces surround it.

### 2.3.5 End game

A Shogi game can end in either a checkmate or a win by draw. Drops limit drastically the possibility of stalemates but the rule still applies and is the same as in International Chess.

## 2.4 Analysis of similarities between variants

This section serves as an analysis of the rules we have just described. The objective is to show how prone to code-reuse these variants are and how it plays an important role in the following chapter which is system design.

The largest similarity is that all three of these variants have the same core game mechanism: two players take turns to make a single move where a piece changes location on or gets removed from a board of x-by-x squares with the goal of checkmating the opponent's leader piece. Another similarity is that a piece can only occupy one square at a time. Captures all operate in the same way even though in shogi the pieces can be dropped back on the board. Promotion is an ability that is present in all three even though it's not explicitly written as a promotion in Xiangqi (the pawn crossing the river). Movement rules are also very similar to each other and are sometimes identical such as the International Chess rook, the Xiangqi rook and the Shogi rook.

If we imagine International Chess as a reference point, then xiangqi is not that much different. It differs in that the pieces in International Chess are free to roam everywhere they want on the board whereas Xiangqi enforces board zones which restrict certain piece movements and allows special things such as pawns getting the ability to move sideways.

Let us proceed in the same manner for Shogi. The largest difference is that pieces that are captured need to be placed on a sideboard. The next is that these pieces can then be dropped back on the board. Promotion is also different in that it isn't something that is obligatory, and Shogi also introduces the idea of zones but for promotion.

The existence of all these already suggests that code re-use would seem a suitable way to implement the game. By simply listing them out a few design ideas seem to surface on their own. The first thing that comes to mind is that a piece is not so different from one variant to the next. If we can manage to bring together all the different attributes into one object, then we will avoid the trouble of having different objects for each variant. Let us attempt this now.

From our analysis, we can infer that a piece has the following attributes:

- Can either be captured or not.
- Has a location on the board.
- Is of a certain type.
- Can act differently according to its location on the board and the number of times it's moved.
- Belongs to a player

Since all variant boards are just aggregates of pieces, our board object will have to capture this information in an array of piece objects with the attributes listed above. The board object will also have to keep track of information about itself such as the number of vertical and horizontal squares it has.

When it comes to gameplay, a generalized function can be called from all board objects and used to generate valid piece moves. This is possible because a piece move is simply an application of rules that dictate where the piece can and can't go. That is to say it is not codependent on its environment.

Here code re-use will not only simplify the code by limiting the number of functions being created which in turn limits the risk of error, but it also helps us in terms of communication as we will see in the design and implementation phase.

## 2.5 Analysis of existing websites

This section is used for the purpose of discussing relevant existing chess websites and what dos and don'ts we can draw from them for our own system. We will also appreciate how different each website is from one another and how difficult it would be for a user to switch from one platform to the next.

For our analysis, let us pick four different websites which appear at the top of the google search engine for the following key words: “play chess online”, “play shogi online”, and “play xiangqi online”. Out of this we get: Lichess.org, Chess.com, Japanesechess.org, Clubxiangqi.com. International Chess has two picks and that is because the design choices between Lichess.org and Chess.com are so different from one another that it is worth looking at both. Moreover, since International Chess is mostly played by native English speakers and my search was in English, the reader needs to keep in mind that there is a strong chance for Clubxiangqi.com and Japanesechess.org not being the best alternative versions available compared to if we were to make a search in mandarin or Japanese. These websites are what we will be basing our simple analysis on.

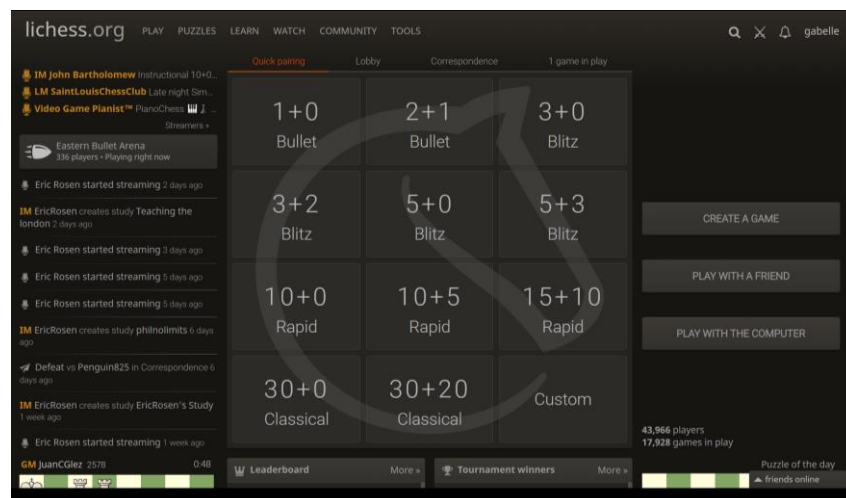


Figure 2.5.1 – screenshot of Lichess.org’s home page



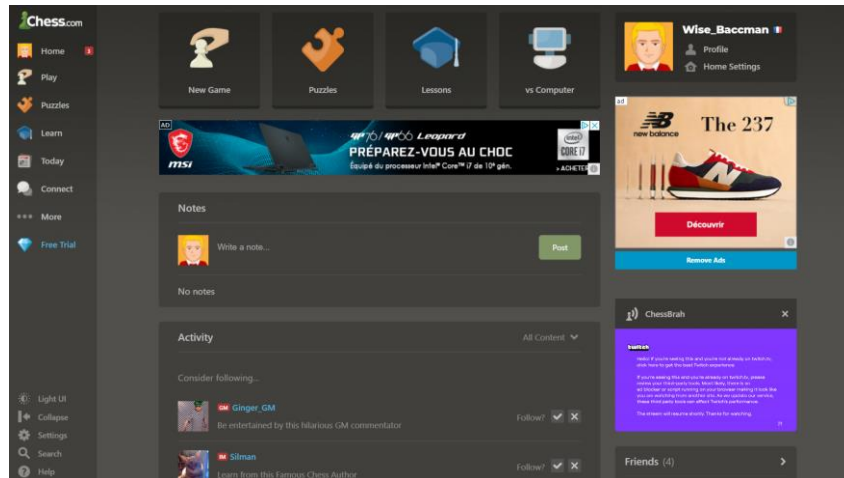


Figure 2.5.2 – screenshot of Chess.com's home page

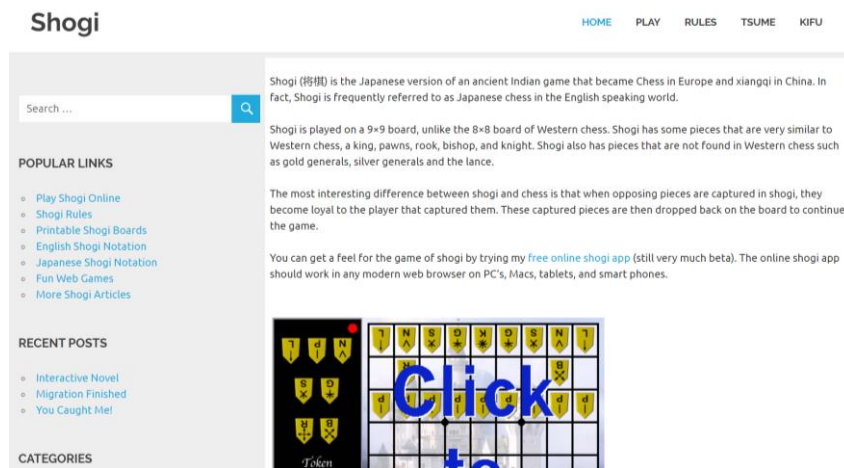


Figure 2.5.3 – screenshot of Japanesechess.org's home page

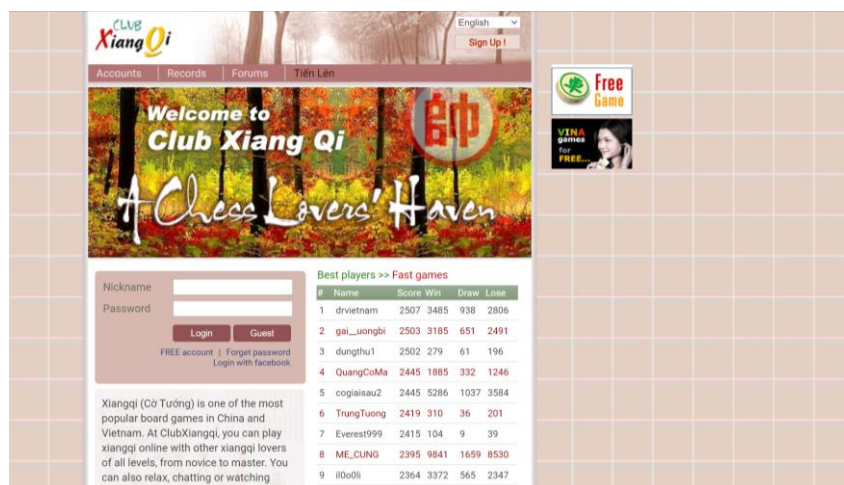


Figure 2.5.4 - screenshot of Clubxiangqi.com's home page

Let us start with the design and accessibility of all websites. Color, imagery and typography are important to communicate messages and evoke emotional response. Japanesechess.org in my opinion has room for improvement in this regard because of odd font choices and distracting background board images. For example, at the bottom of Figure 2.5.3, it seems unfitting to have an image of a castle behind the board while a user is trying to play a game of strategy which demands concentration. On the opposite of the spectrum, we have Lichess.org which is simply appealing to the eye because of beautiful typography and a sleek elegant feel and look.

In terms of navigability, Chess.com is a bit chaotic because it is so not intuitive. Much like Lichess.org, the user's learning curve is high because there is just so much information to process at the same time and buttons are not found intuitively but rather through muscle memory and pattern recognition. Navigation needs to be kept simple.

Visual hierarchy is very relevant particularly when playing a game like chess as the board must be clearly visible and avoid having irrelevant distractions around it. I believe that Clubxiangqi.com has a good attempt at this by popping out a new window when creating a game, establishing a clear focal point as to where the game is taking place. On the other hand, Japanesechess.org has no clear delimitation between a paragraph and the game itself.

From this, I believe that my system should have a home page that is not cluttered and that is as simple as possible. My system should also avoid using multiple fonts and make proper use of style sheets to create a feel to the website that is welcoming and intuitive for newcomers lowering the learning curve. The navigation bar must also be as straightforward as possible to avoid frustration when trying to find a page. Proper use contrast and visual hierarchy will also help reduce learning curve by helping the user know what to focus on.

By doing this analysis the reader can also realize how different all of these websites are and take in the usefulness of a website like the one we are about to create.



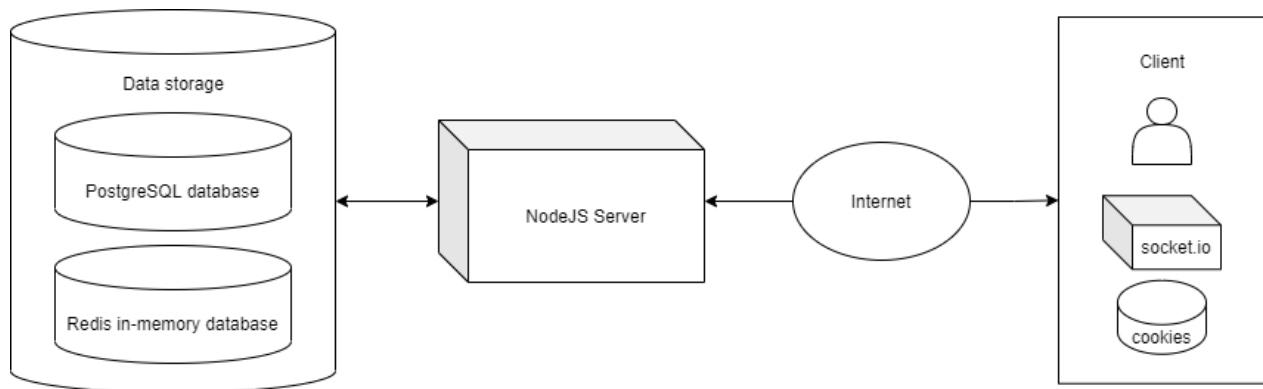
## 3. System Design

### 3.1 Section aims

In the same way that an architect creates the plans for the erection of a building, the system design chapter is just as important to set the system requirements, user classification, system and subsystem architecture, data and database design, input formats, output layouts and processing logic. In other words, it is a way to describe what our application should look like, how it should be structured, how we can evaluate the completeness of it and what steps will be necessary to reach them.

### 3.2 System entities & Attributes

Our aim is to build a website with authentication, and this requires that we create a database, server, session storage for persistency and a client. Let us see how the general design will look like in terms of communication channels between these large entities.



*Figure 3.2.1 – System and attributes communication channels*

As seen in Figure 3.2.1, NodeJS was chosen for the server. The reasons behind this are that it is a cross-platform, reliable, event-driven I/O model with a large community, detailed and useful documentation. It is also known to be performant and easy to operate compared to other servers that use languages such as PHP or Ruby which are more convoluted. NodeJS uses JavaScript which will make data communication straightforward and more consistent since the browser scripts are written in JavaScript.

PostgreSQL is my choice for data storage. It was the logical choice since I needed a free relational database that was quick and easy to set up. Moreover, since PostgreSQL is open source there are many libraries that are available and compatible with NodeJS.

For real-time data communication we will be using sockets, – a socket is defined by an endpoint of a two-way communication link running on the network - and more specifically the JavaScript socket.io library. Sockets work differently to user-driven models and it is necessary for this project as the server needs to send game states to the other player once a move has been made.

### 3.3 Database Design

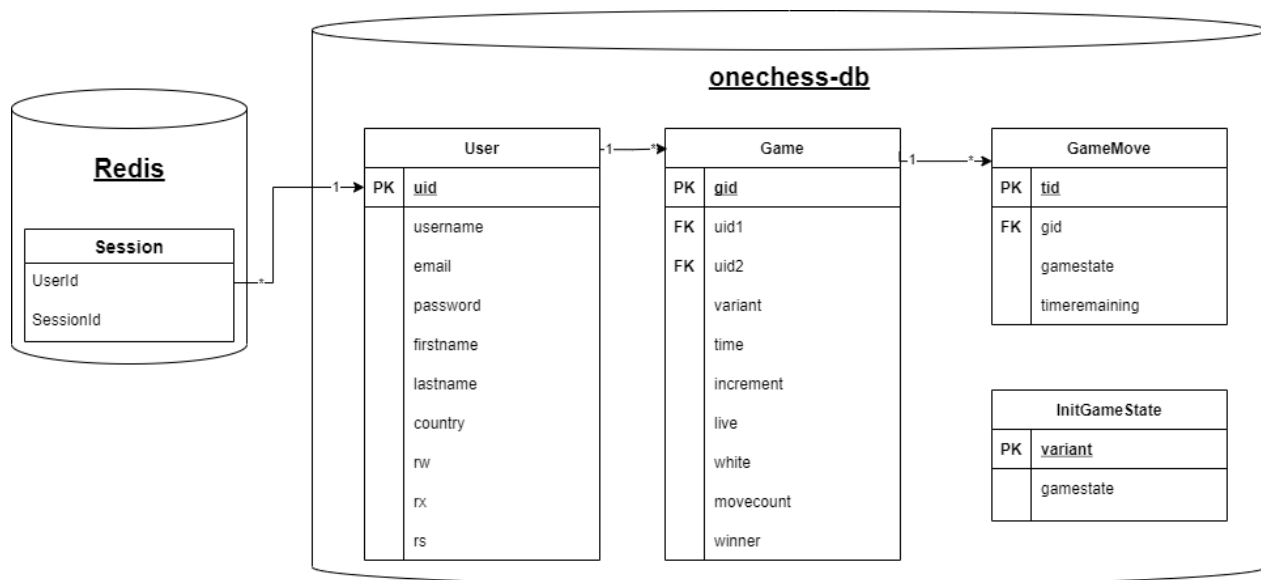
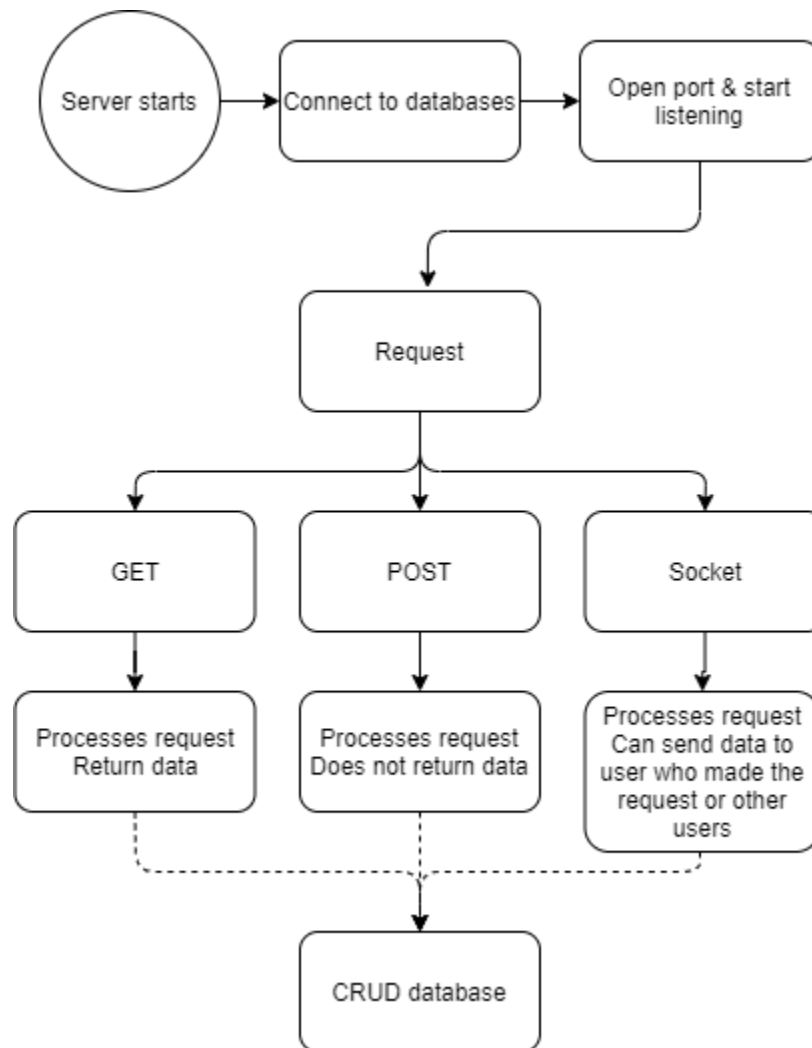


Figure 3.3.1 – database design

Figure 3.3.1 helps us understand what kind of data is necessary for this website and how it will be stored and categorized. The User table is required for authentication services. The Game table is required to have a persistent server that can produce game states whenever necessary. If we did not store game states, then once a client refreshes the page (also resetting the browser memory) the game will be lost and can't be continued. It is necessary to separate the Game data from the User data as one user can have 1 to many games. The same goes for GameMove and Game, a game can have 1 to many moves. The first game state will always be the initial board setup, and this is stored in InitGameState; this is mostly used to avoid redundant data being stored in the Game table. Finally, session storage is imperative to avoid users having to enter their credentials every time they make a request. A Redis server is used to do this. It is an in-memory data store with in-memory key-value pairs most often used for keeping track of sessions. Here the key will be the SessionId which will be stored in a cookie in the user's browser memory and the value is the UserId. Combined with the PostgreSQL our system will have complete authentication.

### 3.4 Flow of Execution

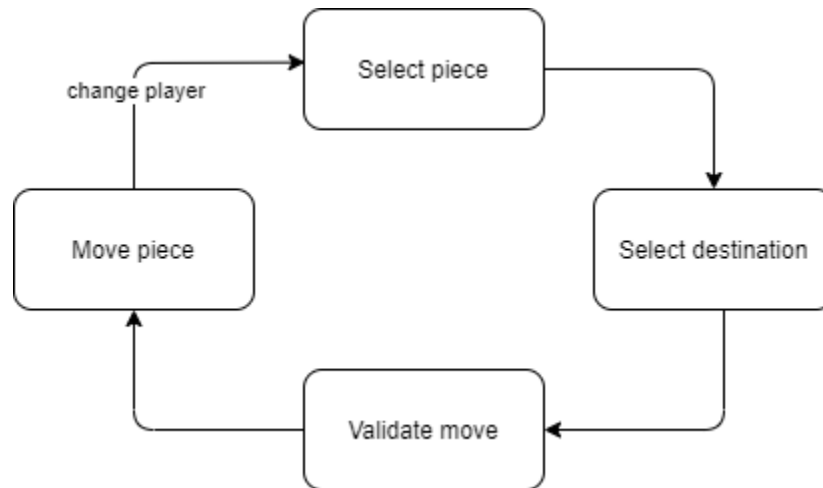
This section is used to suggest certain execution flows that could be implemented for this project. By doing this I hope that the user will be able to understand at least the large picture of how a chess game operates frontend and backend.



*Figure 3.4.1 – server flow*

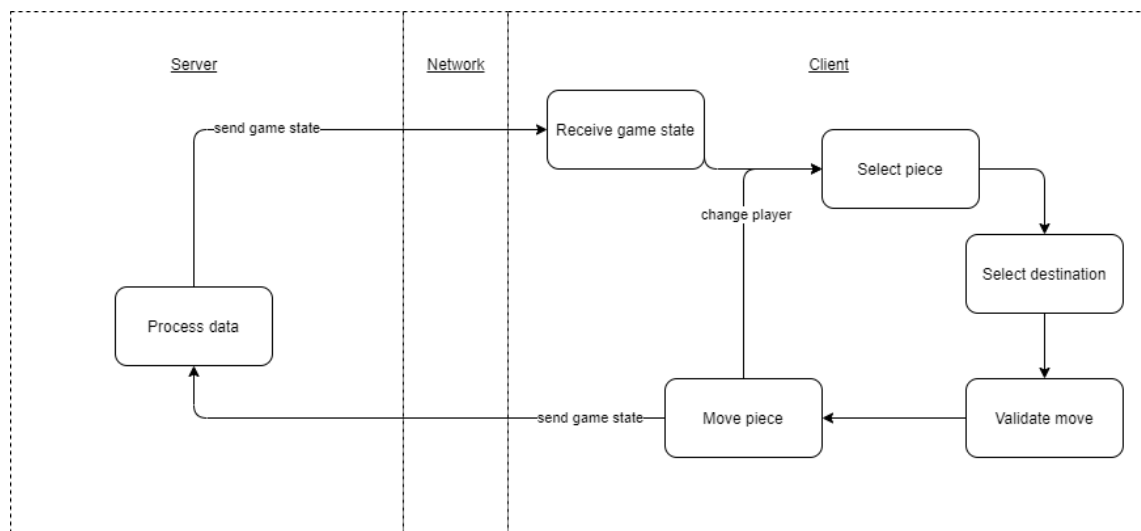
Figure 3.4.1 shows how most of the server's lifecycle is spent waiting for requests. GET and POST are different from the socket requests because the communication between two socket endpoints is continuous. The server is constantly sending a message to the user to see if he is still connected. By doing this, the server ensures that a socket has not been opened for no reason at all. The more requests the server receives the busier and the more congested it becomes. After

receiving a request, it might react in a different way according to the request type but it will most likely send a query to a database, process the data and return values to one or more users.



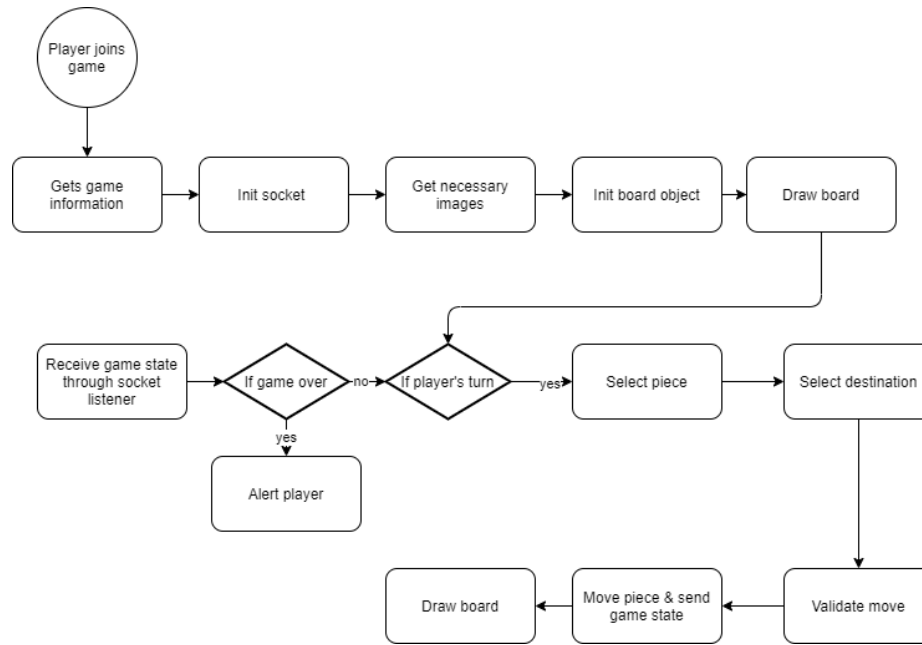
*Figure 3.4.1 – single-player chess flow*

Figure 3.4.1 shows how the flow of an offline single player chess game would look like. It is quite simple and straightforward. However, once we add networking it quickly gets more complicated as the game state is no longer something stored and modified at one location but is multifaceted since it can be modified by multiple users.



*Figure 3.4.2 – two-player online chess flow*

Figure 3.4.2 is a revised flow which shows how, at the end of each turn, the client must send the game state to the server so that it can process, store and transfer the game state to the other player.



*Figure 3.4.3 – client-side game flow*

Figure 3.4.3 takes a closer look at what the flow of execution would like client-side.

### 3.5 Game Data Architecture

Following the analysis of the chess variants in section 2.4, we have mentioned the fact that board objects need to hold information about game pieces. Let us look how this would be structured.



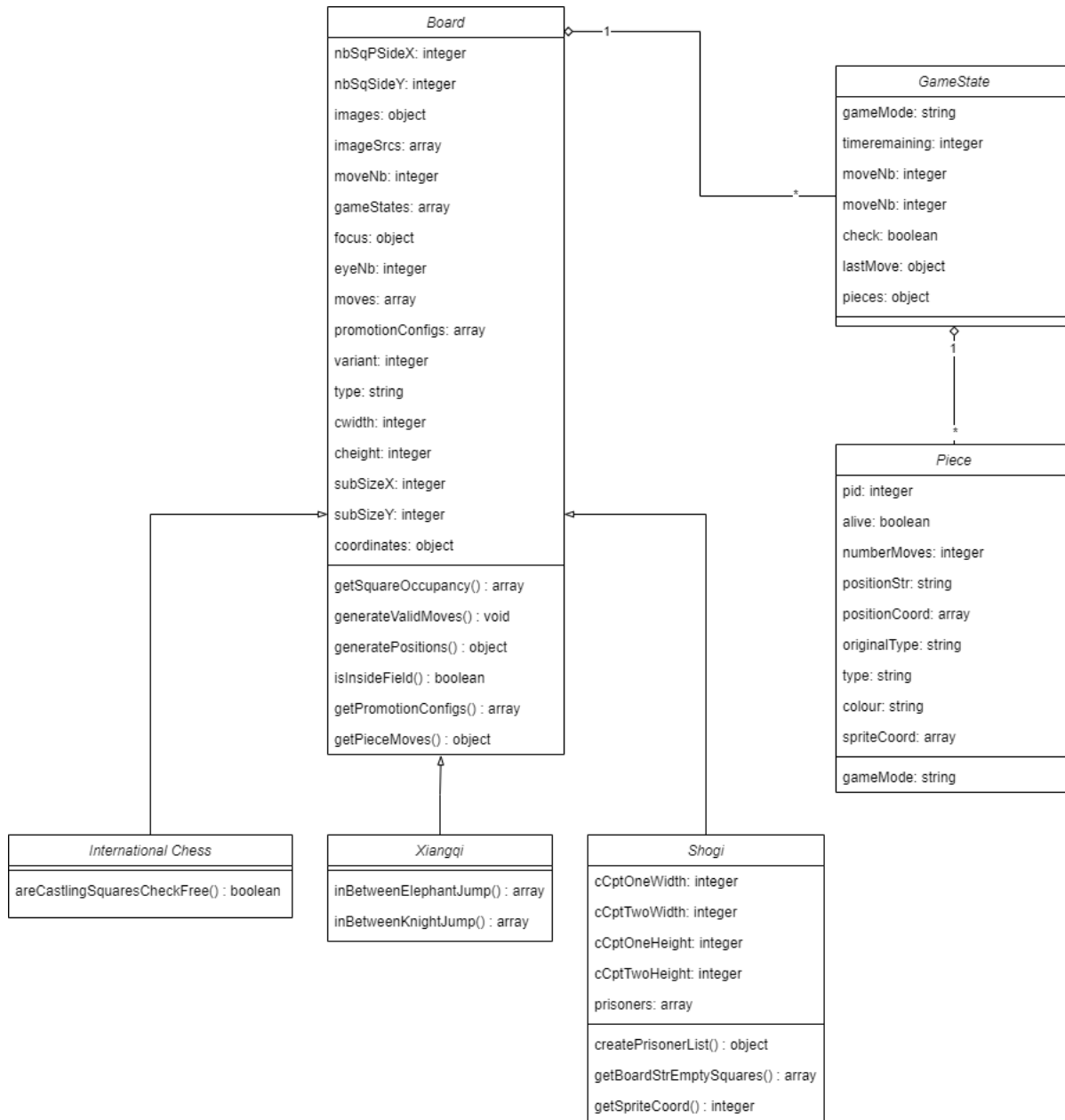


Figure 3.5.1 – class diagram of board and pieces

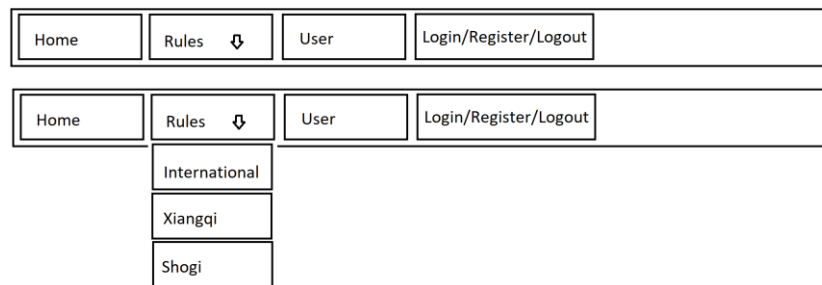
It is important to use inheritance here to connect the variants to the parent class as it provides code re-usability. Instead of writing the same code, again and again, we can simply inherit the properties and methods of the board class into the variants. Notice in Figure 3.5.1 that the three variants do not require too many other functions or attributes than the parent class does. This shows the power of code re-usability for this project. The Piece and Board attributes were made from what we discussed in section 2.4.

In terms of methods and functionality, generateValidMoves will play the most important role. It must be able to iterate through all the pieces on the board and generate valid moves for each one.

The valid moves will be stored in the moves attribute and will be used throughout the program to check whether a piece move is valid or not.

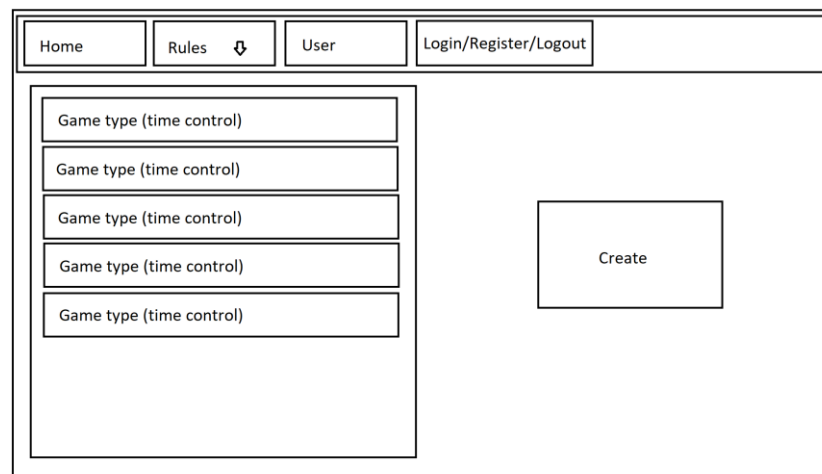
### 3.6 User Interface

Let us take a closer look at what the user will see once on the website. I will be using wireframes to demonstrate this. Wireframes are an important tool for understanding human-computer interaction as it straightens out what the needs of the website are and helps us start to imagine what the underlying technology to support it will be.



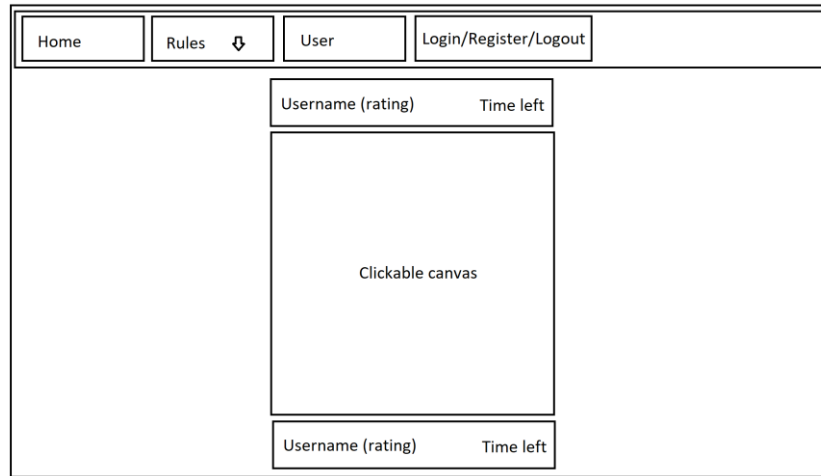
*Figure 3.6.1 – navigation bar wireframe*

Figure 3.6.1 shows a wireframe of the navigation bar that should be available on every page. Upon clicking on a button, the system will react in three different ways. First, it will either redirect the user to the page that corresponds with the button name, this is the case of the Home, User, Register and Login button. Second, it will display other options on the same page, this is the case of the Rules button. Third, it will take some sort of action, this is the case of the Logout button.



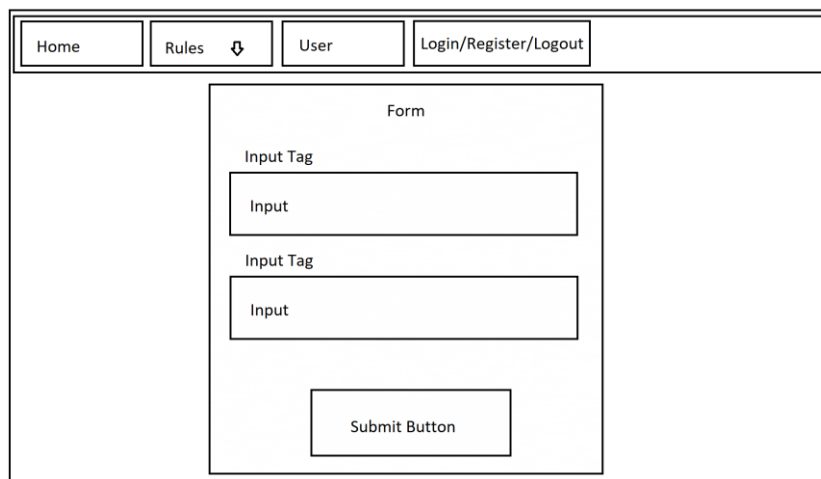
*Figure 3.6.2 – home page wireframe*

The home page user interface, or UI for short, should display games created by other users who are seeking partners in an orderly manner with information about the game. A Create button should also be available on this page as shown in Figure 3.6.2.



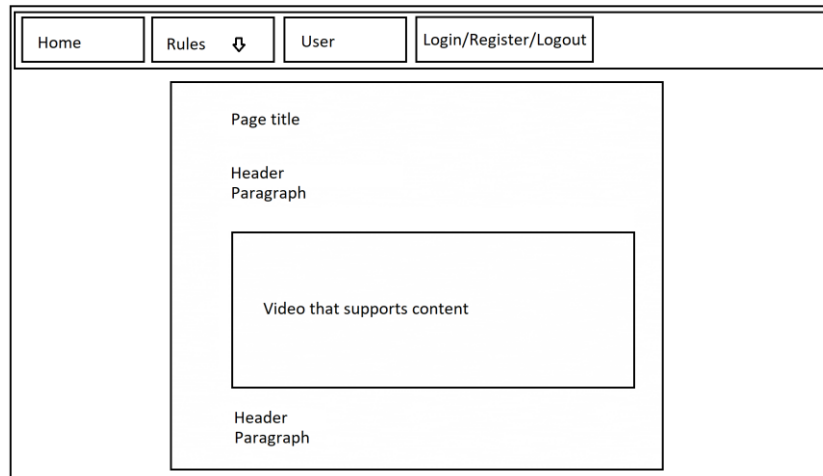
*Figure 3.6.3 – game page wireframe*

The main component for our gameplay interface is the board. The client interacts with the board using clicks to both select the piece and then to drop the piece. When a piece is selected, squares should be highlighted to show valid moves. If a user would like to change perspective by looking at the game from the opponent's point of view the “w” and “b” keys should do just that. The user should also be able to move from through the game states to see what moves have previously been played by using the arrow keys. On either side of the board information about the players should be displayed such as their ranking in the variant and their username. Time control should also be displayed on either side of the board to show how much time is left. The wireframe for this UI is shown in Figure 3.6.3.



*Figure 3.6.4 – form wireframe*

Forms on this website will follow the structure outlined in Figure 3.6.4, this includes the Create, Register and Login page.



*Figure 3.6.5 – article wireframe*

Figure 3.6.5 shows how articles would look like. This wireframe should be used for the Rules pages.

### 3.7 Requirements

By looking over the previous subsections we can develop some requirements that our system should satisfy.

- A user must be able to connect to the server.
- A user must be able to authenticate.
- A user must be able to create a game.
- A user must be able to join a game from the home page.
- The system must be able to generate valid moves for each piece in a game state.
- The system must make the game state visible to the user.
- A user must be able to select a piece by clicking on it.
- A user must be able to view all valid moves highlighted after selecting the piece.
- A user must be able to drop a piece to the destination square by clicking on it.
- The system must be able to transfer the move played by one player to another.
- The system must oblige the player with the white (for International Chess), red (for Xiangqi) or white (for Shogi) pieces to make the first move.
- The system must detect checkmates from a game state and stop the game.
- The rules unique to each variant described in the section 2 must be applied to the moves and outcomes of the games.

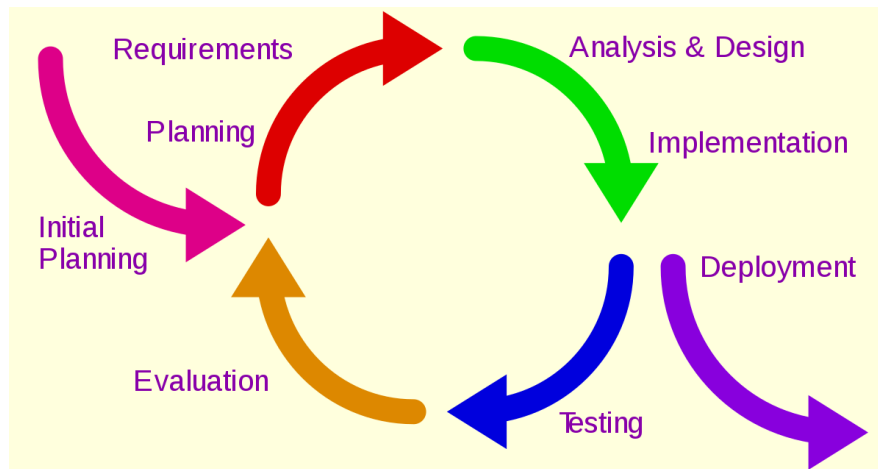


## 4. Implementation

### 4.1 Section Aims

In this section we will be looking at the implementation results as well as the way that we achieved them. To do this we will firstly discuss the development methodology utilized and why it was chosen. Secondly, we will look at the key implementation stages and everything encompassing them. Finally, we will look at the results and compare them to how we had originally conceived them during the system design and evaluate how far or how close we ended up being.

### 4.2 Development Methodology



*Figure 4.2.1 – model showing the incremental development methodology*

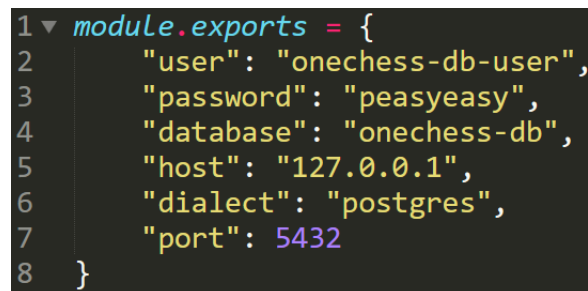
The development process used for this project is incremental development. Incremental development is a methodology where small features build on top of each other to reach a certain goal. It is different from the waterfall model approach and other types of methodologies by which a working system only becomes available in the later phases of the project. It starts with a very basic working system which is improved step by step. Figure 4.2.1 shows in a visual way how this is organized. This methodology suited me perfectly because at the start of the project I struggled to articulate the system specifications. It was only after a few months into the project that I had the knowledge and expertise to confidently know which path I had to follow and where the project was heading.

## 4.3 Key Implementation Stages

### 4.3.1 Basic Server-Client

The first step was to figure out how to run a simple NodeJS server. NodeJS comes preinstalled with npm, short for node package manager, which is an extremely useful tool for installing packages from the command line. It is like pip for python or sudo install in Linux. Downloading and using libraries such as Express simplified drastically this process to the point where five lines of code could run a server.

Express sits on top of NodeJS and it uses the http object to initialize itself. The variable is generally called app and requests are caught inside of handlers such as: app.get("/index.html"...). A handler is used to catch a request and the logic inside of the method is used to process the input data and send a response.



```
1 module.exports = {
2   "user": "onechess-db-user",
3   "password": "peasyeasy",
4   "database": "onechess-db",
5   "host": "127.0.0.1",
6   "dialect": "postgres",
7   "port": 5432
8 }
```

Figure 4.3.1.1 – image showing the configuration file for connecting the server with PostgreSQL



```
75 // Psql co & creating tables
76 const pool = new Pool(PSQLConfig);
77
78 pool.query(tableQueries.Game, (err, res) =>
79 {
80   if (err) {
81     console.log(err);
82   } else {
83     console.log(res.command + " Table Game");
84   }
85 });
86
```

Figure 4.3.1.2 – image showing a simple SQL query being sent to PostgreSQL via the pool which is created from the configuration file shown in Figure 4.3.1.1

The next step was trying to figure out how to connect the server to PostgreSQL. Once more, NodeJS has a great library for this called “pg”. All that was required is a config file like the one

in Figure 4.3.1.1 and the creation of an object called a Pool as shown in Figure 4.3.1.2. The same logic was applied for the Redis session store and by using it as a middleware this will be useful to implement the authentication system.

At this point we have a running server that can connect to a database. The next logical step was to send something useful back to the client. Using pug, a html template engine, we were able to create all our views which can be seen in section 4.3. Pug files could be rendered when the page was requested, and this would avoid having to waste another round-trip-time.

### 4.3.2 Server-Client Communication

Our server is almost done, the last piece of logic that is not mentioned are sockets. Socket.io, the most popular socket library for NodeJS was used to accomplish this. It sits on top of the http server and it detects when a request comes in. As explained beforehand, the server will then send continuous streams of packets at regular intervals to the client to see if he is still active else the connection is stopped immediately with a timeout. The socket.io JavaScript library needs to be included inside of the client script or else this does not work. This library includes useful methods that enable both the server and the client to establish this communication path.

We have not yet talked about the kind of data that is being sent between the client and the server so let us do that in this paragraph. The most peculiar thing to look at is the actual game data itself as other requests have nothing out of the ordinary about them; they are plain GET and POST requests and act just as you would expect. For this, all takes place inside of the game.js file that is executed in the frontend and app.js that is the main server file.

```
{
  "gameMode": "international",
  "timerRemaining": null,
  "moveNb": 0,
  "check": false,
  "lastMove": {
    "piece": null,
    "originStr": null,
    "originCoord": null,
    "destinationStr": null,
    "destinationCoord": null,
    "pieces": [
      {
        "pid": 0,
        "alive": true,
        "numberMoves": 0,
        "positionStr": "a1",
        "positionCoord": [0, 0],
        "originalType": "rook",
        "type": "rook",
        "colour": "white",
        "spriteCoord": 8,
      },
      {
        "pid": 1,
        "alive": true,
        "numberMoves": 0,
        "positionStr": "b1",
        "positionCoord": [1, 0],
        "originalType": "knight",
        "type": "knight",
        "colour": "white",
        "spriteCoord": 6,
      },
      {
        "pid": 2,
        "alive": true,
        "numberMoves": 0,
        "positionStr": "c1",
        "positionCoord": [2, 0],
        "originalType": "bishop",
        "type": "bishop",
        "colour": "white",
        "spriteCoord": 4,
      },
      {
        "pid": 3,
        "alive": true,
        "numberMoves": 0,
        "positionStr": "d1",
        "positionCoord": [3, 0],
        "originalType": "queen",
        "type": "queen",
        "colour": "white",
        "spriteCoord": 2,
      },
      {
        "pid": 4,
        "alive": true,
        "numberMoves": 0,
        "positionStr": "e1",
        "positionCoord": [4, 0],
        "originalType": "king",
        "type": "king",
        "colour": "white",
        "spriteCoord": 0,
      },
      {
        "pid": 5,
        "alive": true,
        "numberMoves": 0,
        "positionStr": "f1",
        "positionCoord": [5, 0],
        "originalType": "bishop",
        "type": "bishop",
        "colour": "white",
        "spriteCoord": 4,
      },
      {
        "pid": 6,
        "alive": true,
        "numberMoves": 0,
        "positionStr": "g1",
        "positionCoord": [6, 0],
        "originalType": "knight",
        "type": "knight",
        "colour": "white",
        "spriteCoord": 6,
      },
      {
        "pid": 7,
        "alive": true,
        "numberMoves": 0,
        "positionStr": "h1",
        "positionCoord": [7, 0],
        "originalType": "rook",
        "type": "rook",
        "colour": "white",
        "spriteCoord": 8,
      },
      {
        "pid": 8,
        "alive": true,
        "numberMoves": 0,
        "positionStr": "a2",
        "positionCoord": [0, 1],
        "originalType": "pawn",
        "type": "pawn",
        "colour": "white",
        "spriteCoord": 10,
      },
      {
        "pid": 9,
        "alive": true,
        "numberMoves": 0,
        "positionStr": "b2",
        "positionCoord": [1, 1],
        "originalType": "pawn",
        "type": "pawn",
        "colour": "white",
        "spriteCoord": 10,
      },
      {
        "pid": 10,
        "alive": true,
        "numberMoves": 0,
        "positionStr": "c2",
        "positionCoord": [2, 1],
        "originalType": "pawn",
        "type": "pawn",
        "colour": "white",
        "spriteCoord": 10,
      },
      {
        "pid": 11,
        "alive": true,
        "numberMoves": 0,
        "positionStr": "d2",
        "positionCoord": [3, 1],
        "originalType": "pawn",
        "type": "pawn",
        "colour": "white",
        "spriteCoord": 10,
      },
      {
        "pid": 12,
        "alive": true,
        "numberMoves": 0,
        "positionStr": "e2",
        "positionCoord": [4, 1],
        "originalType": "pawn",
        "type": "pawn",
        "colour": "white",
        "spriteCoord": 10,
      },
      {
        "pid": 13,
        "alive": true,
        "numberMoves": 0,
        "positionStr": "f2",
        "positionCoord": [5, 1],
        "originalType": "pawn",
        "type": "pawn",
        "colour": "white",
        "spriteCoord": 10,
      },
      {
        "pid": 14,
        "alive": true,
        "numberMoves": 0,
        "positionStr": "g2",
        "positionCoord": [6, 1],
        "originalType": "pawn",
        "type": "pawn",
        "colour": "white",
        "spriteCoord": 10,
      },
      {
        "pid": 15,
        "alive": true,
        "numberMoves": 0,
        "positionStr": "h2",
        "positionCoord": [7, 1],
        "originalType": "pawn",
        "type": "pawn",
        "colour": "white",
        "spriteCoord": 10,
      },
      {
        "pid": 16,
        "alive": true,
        "numberMoves": 0,
        "positionStr": "a8",
        "positionCoord": [0, 7],
        "originalType": "rook",
        "type": "rook",
        "colour": "black",
        "spriteCoord": 9,
      },
      {
        "pid": 17,
        "alive": true,
        "numberMoves": 0,
        "positionStr": "b8",
        "positionCoord": [1, 7],
        "originalType": "knight",
        "type": "knight",
        "colour": "black",
        "spriteCoord": 7,
      },
      {
        "pid": 18,
        "alive": true,
        "numberMoves": 0,
        "positionStr": "c8",
        "positionCoord": [2, 7],
        "originalType": "bishop",
        "type": "bishop",
        "colour": "black",
        "spriteCoord": 5,
      },
      {
        "pid": 19,
        "alive": true,
        "numberMoves": 0,
        "positionStr": "d8",
        "positionCoord": [3, 7],
        "originalType": "queen",
        "type": "queen",
        "colour": "black",
        "spriteCoord": 3,
      },
      {
        "pid": 20,
        "alive": true,
        "numberMoves": 0,
        "positionStr": "e8",
        "positionCoord": [4, 7],
        "originalType": "king",
        "type": "king",
        "colour": "black",
        "spriteCoord": 1,
      },
      {
        "pid": 21,
        "alive": true,
        "numberMoves": 0,
        "positionStr": "f8",
        "positionCoord": [5, 7],
        "originalType": "bishop",
        "type": "bishop",
        "colour": "black",
        "spriteCoord": 5,
      },
      {
        "pid": 22,
        "alive": true,
        "numberMoves": 0,
        "positionStr": "g8",
        "positionCoord": [6, 7],
        "originalType": "knight",
        "type": "knight",
        "colour": "black",
        "spriteCoord": 7,
      },
      {
        "pid": 23,
        "alive": true,
        "numberMoves": 0,
        "positionStr": "h8",
        "positionCoord": [7, 7],
        "originalType": "rook",
        "type": "rook",
        "colour": "black",
        "spriteCoord": 9,
      },
      {
        "pid": 24,
        "alive": true,
        "numberMoves": 0,
        "positionStr": "a7",
        "positionCoord": [0, 6],
        "originalType": "pawn",
        "type": "pawn",
        "colour": "black",
        "spriteCoord": 11,
      },
      {
        "pid": 25,
        "alive": true,
        "numberMoves": 0,
        "positionStr": "b7",
        "positionCoord": [1, 6],
        "originalType": "pawn",
        "type": "pawn",
        "colour": "black",
        "spriteCoord": 11,
      },
      {
        "pid": 26,
        "alive": true,
        "numberMoves": 0,
        "positionStr": "c7",
        "positionCoord": [2, 6],
        "originalType": "pawn",
        "type": "pawn",
        "colour": "black",
        "spriteCoord": 11,
      },
      {
        "pid": 27,
        "alive": true,
        "numberMoves": 0,
        "positionStr": "d7",
        "positionCoord": [3, 6],
        "originalType": "pawn",
        "type": "pawn",
        "colour": "black",
        "spriteCoord": 11,
      },
      {
        "pid": 28,
        "alive": true,
        "numberMoves": 0,
        "positionStr": "e7",
        "positionCoord": [4, 6],
        "originalType": "pawn",
        "type": "pawn",
        "colour": "black",
        "spriteCoord": 11,
      },
      {
        "pid": 29,
        "alive": true,
        "numberMoves": 0,
        "positionStr": "f7",
        "positionCoord": [5, 6],
        "originalType": "pawn",
        "type": "pawn",
        "colour": "black",
        "spriteCoord": 11,
      },
      {
        "pid": 30,
        "alive": true,
        "numberMoves": 0,
        "positionStr": "g7",
        "positionCoord": [6, 6],
        "originalType": "pawn",
        "type": "pawn",
        "colour": "black",
        "spriteCoord": 11,
      },
      {
        "pid": 31,
        "alive": true,
        "numberMoves": 0,
        "positionStr": "h7",
        "positionCoord": [7, 6],
        "originalType": "pawn",
        "type": "pawn",
        "colour": "black",
        "spriteCoord": 11,
      }
    ]
  }
}
```

Figure 4.3.2.1 – screenshot showing the initial game state for international chess



When two players join a game of chess the server sends to both players the initial game state which can be seen in Figure 4.3.2.1. The structure is very similar to what we'd expected as shown from Figure 3.5.1. The first red section being information about the board itself and the blue section containing information about all pieces on the board. This structure is identical whether it Xiangqi, International Chess or Shogi.

## 4.4 System in Operation

Looking back at section 3.7, let's look at the implementation of these requirements.

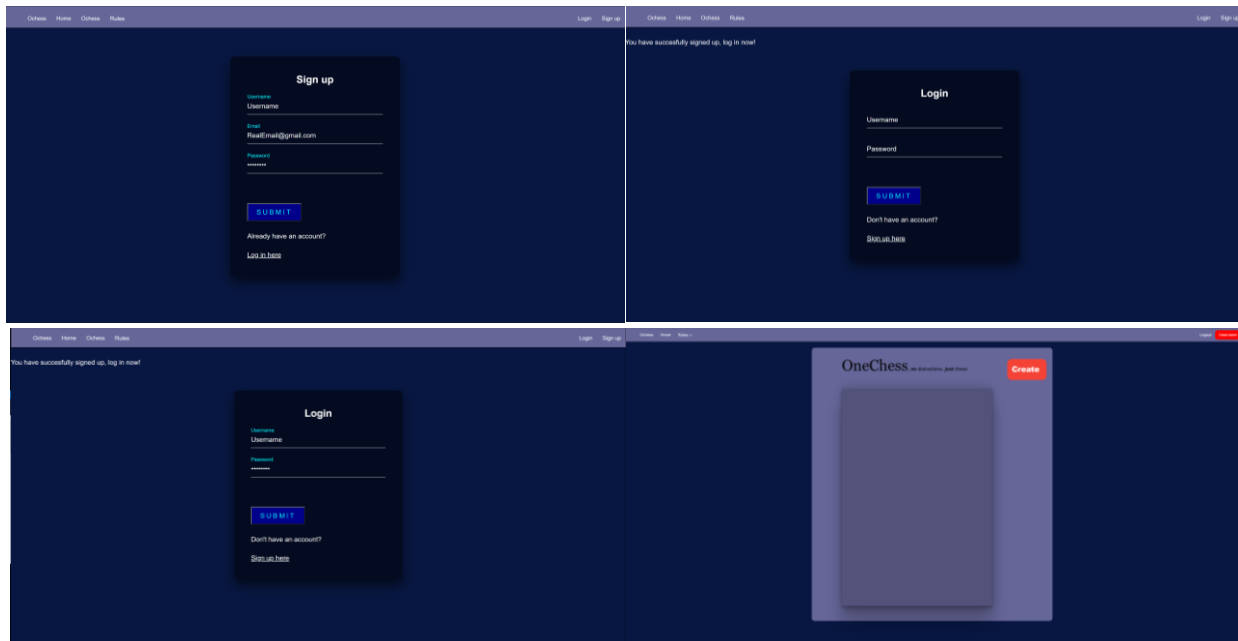


Figure 4.4.1 – screenshots showing a user creating an account and logging in

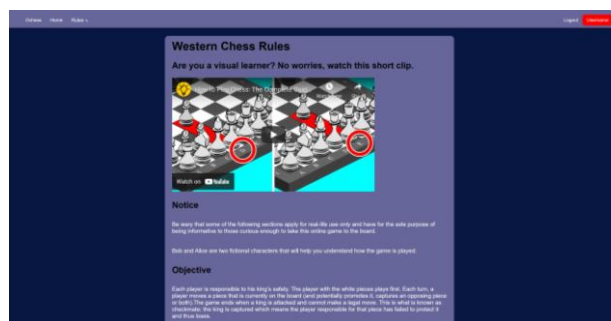


Figure 4.4.2 – screenshot showing the rules page with imbedded video



Figure 4.4.3 – screenshot showing the user's page

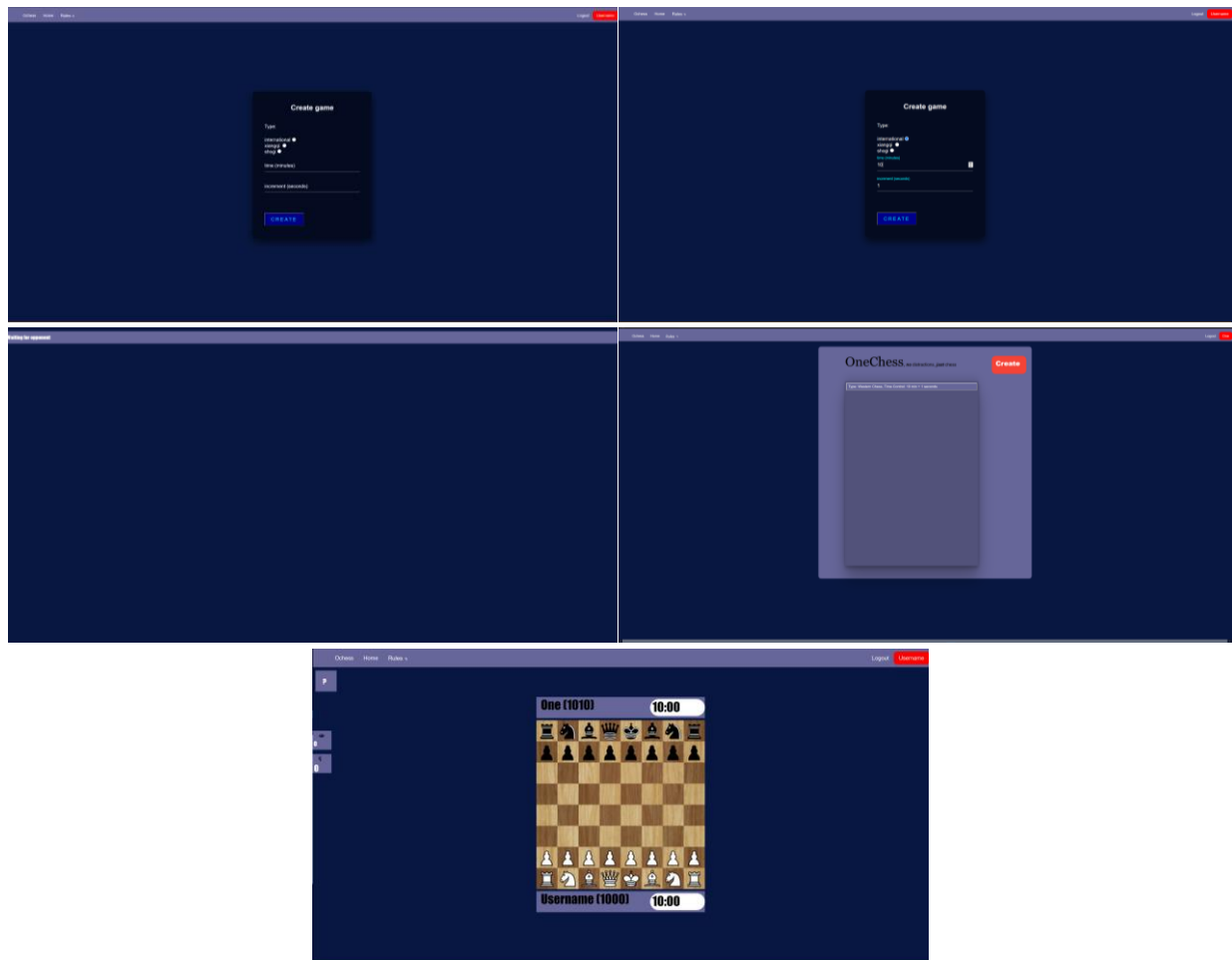


Figure 4.4.4 – screenshots showing a user creating a game and another user joining the game



Figure 4.4.5 – screenshot showing that the first move in International Chess is played by white



Figure 4.4.6 – screenshots showing the highlights and result of a capture

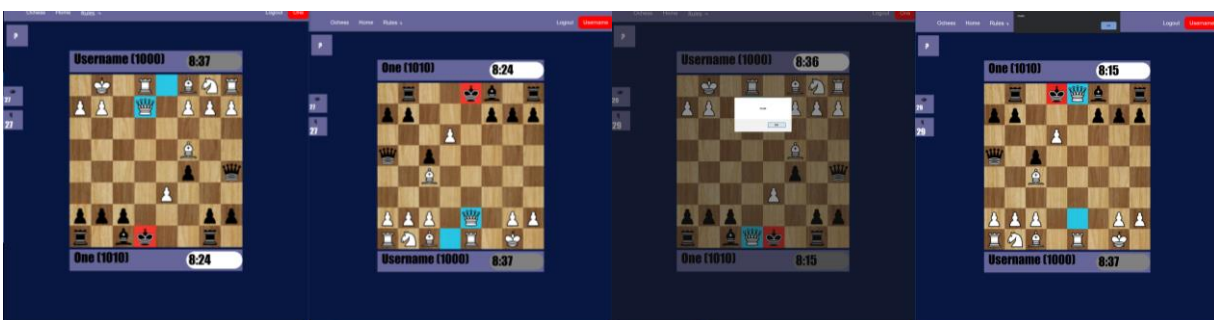


Figure 4.4.7 – screenshots showing how a checkmate is delivered



Figure 4.4.8 – screenshots showing how a Shogi pawn is dropped on the board

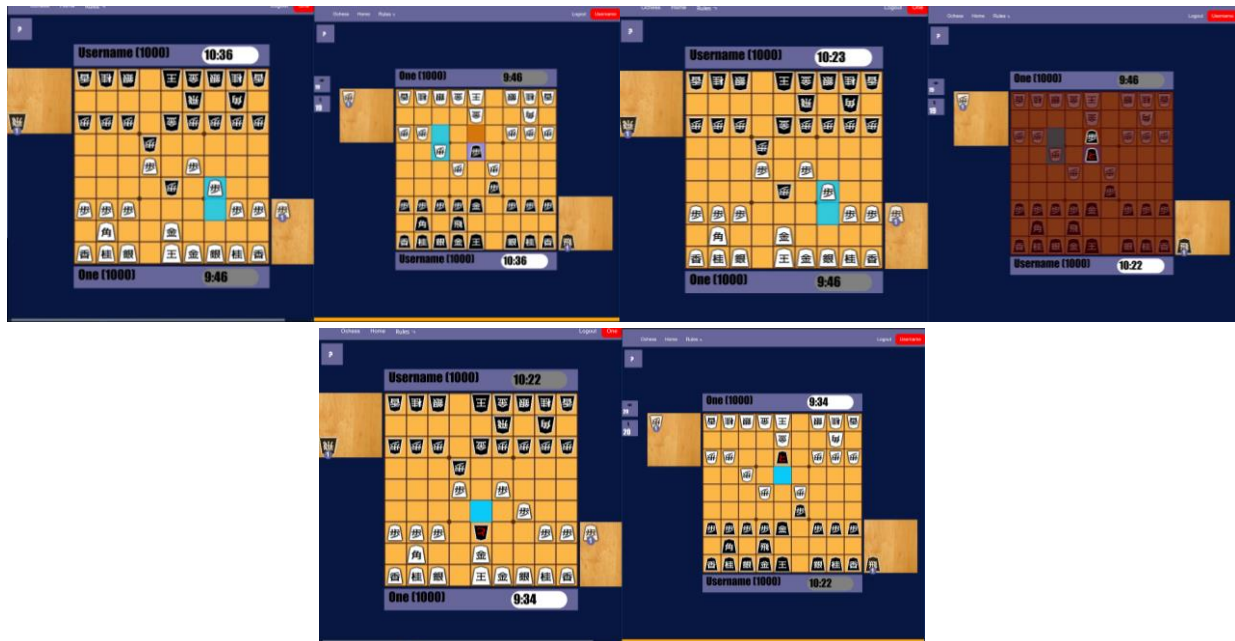


Figure 4.4.9 – screenshots showing a Shogi pawn promoting in the promotion zone

## 5 CONCLUSION

Looking back at the aims we set at the beginning of this report, one could say that the aims were met since we have managed to create a website where all three variants were implemented. Code re-use was applied effectively and greatly simplified the overall structure of the system. I truly recommend developers to apply this approach as it will break a project's duration in half and it will also provide great deal of relief when the project is wearing you down and you find a method that solves just the problem you were having.

A functionality that I have not explicitly mentioned in this report is the deployment of this application on a real domain with my raspberry pi. A large part of me working on this functionality was spent researching how to connect a public domain to my private machine. I was starting from scratch and wrote all my progress in notepad, scribbling down every interesting piece of information that I found online. I used GoDaddy to get the onechess.org domain and used the A record to point to my IP address which is straightforward and intuitive. With my ISP, hosting isn't serviced by default and I was forced to contact them so that they could let me host a website on my private IP. This took a few weeks and when they got it done it wasn't a job done. I still had a few tasks to accomplish. One of them was port forwarding and the other was setting up a reverse proxy. Both took lots of time because I had no clue what I was doing but I was enthralled by the process and pushed forward. I ended finding out that doing port forwarding is just as simple as connecting to your router, most often using 192.168.0.1, and changing the configuration so that if a request has certain characteristics then it should forward it to my private IP, which is 127.0.0.27. Since NodeJS runs on port 3000 and I wanted users to call onechess.org instead of onechess.org:3000 so had to reverse proxy port 80 requests into port 3000 requests using nginx which proved more difficult than expected considering it's just fives being added to a configuration file. Pushing it to my raspberry pi was the simplest task in this added functionality and it only took one afternoon. The raspberry pi had to be accessed via SSH, so I had to change its settings to change the default desktop bootup into a command line interface bootup. Putty and a tool called pscp was then used to transfer the zip folder inside of it and then it was just as easy as extracting it and flipping the switch on. It was really fun side project and a great learning experience.

I am satisfied with how this system turned out. Its design is very solid but there are things that could be improved, and this paragraph lists them out. Resigning a game or offering a draw should be added to the gameplay. A user should be able to see a list of all the games that they have played for self-analysis. The user interface was simple, but the design feels off and needs improving.

This project has taught me a lot about the reality of programming large solo projects. Coming up with the ideas and concepts of what something will look like is simple but putting these things into practice is another thing. It is slow, brutal, and unforgiving. Programming is not a marathon; it is a hurdle race where you fall every step of the way and the only way to get passed an issue is

to contort your body and mind to circumvent it. However, the feeling after solving an issue is worth the trouble because you feel a sense of purpose and joy like none other.

## **Acknowledgments**

This endeavor was only possible thanks to Paul Dempster, my supervisor during this project.  
Thank you for your support!

## **Bibliography**

### **Image board tool for the International Chess tutorial**

[chessboardimage.com](http://chessboardimage.com).

### **Image board tool for the Xiangqi tutorial**

[pychess.org/editor/xiangqi](http://pychess.org/editor/xiangqi) courtesy of Bajusz Tamás.

### **Creating design diagrams**

[app.diagrams.net](http://app.diagrams.net) also known as draw.io

### **Diagram from Wikipedia article, “Iterative and Incremental Development”**

[en.wikipedia.org/wiki/Iterative\\_and\\_incremental\\_development](http://en.wikipedia.org/wiki/Iterative_and_incremental_development)

### **Images for Xiangqi tutorial, piece movement**

[ancientchess.com/page/play-shogi.htm](http://ancientchess.com/page/play-shogi.htm)