1/1/2024

Varun Vivek Nayak

Grammar School at Leeds

A-Level Computer Science NEA



[**Analysis** 4](#_Toc158842889)

[**Purpose** 4](#_Toc158842890)

[The solution to the problem 5](#_Toc158842891)

[**Conversation** 5](#_Toc158842892)

[Screenshots of conversation 8](#_Toc158842893)

[**Research** 10](#_Toc158842894)

[**Game notation** 11](#_Toc158842895)

[**Rules** 11](#_Toc158842896)

[General rules 13](#_Toc158842897)

[Additional rules as part of my variant Big Bang Chess 13](#_Toc158842898)

[**Proof of concept** 14](#_Toc158842899)

[**Idea for a new chess variant** 14](#_Toc158842900)

[**Improving my knowledge foundation of Python Programming** 15](#_Toc158842901)

[15](#_Toc158842902)

[**Decomposition of the problem** 16](#_Toc158842903)

[Use case diagram 17](#_Toc158842904)

[**APIs** 17](#_Toc158842905)

[APIs in the context of this project 18](#_Toc158842906)

[Modelling the API fetch 18](#_Toc158842907)

[**Parsing** 19](#_Toc158842908)

[**Current Systems** 20](#_Toc158842909)

[FICS 20](#_Toc158842910)

[chess24 20](#_Toc158842911)

[**Requirements** 21](#_Toc158842912)

[Questionnaire 21](#_Toc158842913)

[Objectives 25](#_Toc158842914)

[**The main algorithms** 29](#_Toc158842915)

[**Design** 31](#_Toc158842916)

[User interface – how it looks to the user 31](#_Toc158842917)

[The games components 33](#_Toc158842918)

[Code structure employed for each component 33](#_Toc158842919)

[**Structure diagram for the games components** 34](#_Toc158842920)

[Big Bang Variant 34](#_Toc158842921)

[Standard Variant 34](#_Toc158842922)

[**Data Flow Diagram** 34](#_Toc158842923)

[**System Flowchart** 35](#_Toc158842924)

[Standard Chess Variant Flowchart 36](#_Toc158842925)

[Big Bang Variant Flowchart 37](#_Toc158842926)

[View Lichess Stats Flowchart 38](#_Toc158842927)

[**UML Diagrams** 39](#_Toc158842928)

[Big Bang Variant 39](#_Toc158842929)

[Running the Big Bang Variant Game 40](#_Toc158842930)

[UML for my user interface 41](#_Toc158842931)

[**Database design** 41](#_Toc158842932)

[The tables 42](#_Toc158842933)

[PasswordDetails Table 42](#_Toc158842934)

[IDDetails Table 42](#_Toc158842935)

[UsernameDetails Table 42](#_Toc158842936)

[**E-R diagrams for the above system** 42](#_Toc158842937)

[The Data Definition Language breakdown of the database 44](#_Toc158842938)

[**Key algorithms** 44](#_Toc158842939)

[Checkmate checker pseudocode 44](#_Toc158842940)

[Checkmate checker flowchart 45](#_Toc158842941)

[Draw board algorithm 46](#_Toc158842942)

[Fetch move algorithm pseudocode 46](#_Toc158842943)

[**Fetch Move algorithm flowchart** 47](#_Toc158842944)

[**Data structures** 47](#_Toc158842945)

[Stack 47](#_Toc158842946)

[**Implementation** 49](#_Toc158842947)

[**Calling parameterised Web service APIs and parsing JSON to service a complex client server model** 49](#_Toc158842948)

[**Cross table SQL** 49](#_Toc158842949)

[**SQL tables** 49](#_Toc158842950)

[**SQL functions** 49](#_Toc158842951)

[**List operations** 49](#_Toc158842952)

[**Recursive algorithms** 49](#_Toc158842953)

[**Hashing** 49](#_Toc158842954)

[**Stack/Queue operations** 49](#_Toc158842955)

[**Dynamic generation of objects based on complex user-defined use of OOP model** 50](#_Toc158842956)

[Inheritance 50](#_Toc158842957)

[Polymorphism 50](#_Toc158842958)

[Interfaces 50](#_Toc158842959)

[Composition 50](#_Toc158842960)

[Classes 51](#_Toc158842961)

[**Writing and reading from text files** 51](#_Toc158842962)

[**Dictionaries** 51](#_Toc158842963)

[**Complex Mathematical model** 51](#_Toc158842964)

[**Single Dimension Array** 51](#_Toc158842965)

[**The code** 52](#_Toc158842966)

[**home.py** 52](#_Toc158842967)

[**ViewLichessStats.py** 84](#_Toc158842968)

[**bigbangchessapp.py** 88](#_Toc158842969)

[**boardclass.py** 97](#_Toc158842970)

[**piececlass.py** 104](#_Toc158842971)

[**serpentclass.py** 104](#_Toc158842972)

[**empressclass.py** 105](#_Toc158842973)

[**kingclass.py** 106](#_Toc158842974)

[**queenclass.py** 107](#_Toc158842975)

[**rookclass.py** 107](#_Toc158842976)

[**bishopclass.py** 108](#_Toc158842977)

[**pawnclass.py** 109](#_Toc158842978)

[**stack.py** 110](#_Toc158842979)

[**Data files** 111](#_Toc158842980)

[**blitzratings.txt** 111](#_Toc158842981)

[**bulletratings.txt** 111](#_Toc158842982)

[**rapidratings.txt** 112](#_Toc158842983)

[**classicalratings.txt** 112](#_Toc158842984)

[**General Implementation evaluation of my end user’s objectives from my own perspective** 112](#_Toc158842985)

[**Testing** 113](#_Toc158842986)

[Post-development testing 113](#_Toc158842987)

[**Evaluation** 114](#_Toc158842988)

[**Requirement 1** 114](#_Toc158842989)

[**Requirement 2** 115](#_Toc158842990)

[**Requirement 3** 116](#_Toc158842991)

[**Requirement 4** 118](#_Toc158842992)

[**Requirement 5** 119](#_Toc158842993)

[**Requirement 6** 120](#_Toc158842994)

[**Requirement 7** 120](#_Toc158842995)

[**Requirement 8** 121](#_Toc158842996)

[**My end-user’s thoughts** 122](#_Toc158842997)

[The visual display 122](#_Toc158842998)

[Piece functionalities 123](#_Toc158842999)

[Navigation 123](#_Toc158843000)

[Making a move 123](#_Toc158843001)

[Ending the game 124](#_Toc158843002)

[Drawing the ratings graph and fetching the PGN notation of my games 124](#_Toc158843003)

[**My comments about the end-user’s feedback** 124](#_Toc158843004)

[The visual display 124](#_Toc158843005)

[Piece functionalities 124](#_Toc158843006)

[Navigation 125](#_Toc158843007)

[Making a move 125](#_Toc158843008)

[Ending the game 125](#_Toc158843009)

[Drawing the ratings graph and fetching the PGN notation of my games 125](#_Toc158843010)

[**Bibliography** 126](#_Toc158843011)

# **Analysis**

## Lichess Front-End Database Server is down and a new server is under construction for the moment. As of 2021-04-13 Tuesday 4.24 AM CEST : r/ lichessA person working on a computer Description automatically generated**Purpose**

The end-user for my A level Computer Science NEA is Vishnu Vadlamani, who is one of my friends and plays chess for Leeds Junior Chess Club. My end user is an avid chess player who frequently plays competitively in tournaments. The main problem for him is that for him to be able to access his studies and preparation for these tournaments, he needs an Internet connection and mentioned that the recent boom in chess due to the COVID pandemic has meant that at peak times of demand, he cannot log in and play games as practice. As a result, when I contacted him to see whether they had an issue he wished for me to solve for him, this was brought to my attention, as shown by the conversation which I have included on the next page. Additionally, he mentioned that there are often several security checks that are carried out in addition to the login process on chess websites, which mean that it often takes up quite a bit of his valuable time. Furthermore, with these online platforms that currently exist (he mentioned lichess.org and chess.com in particular), it is becoming increasingly easier for his opponents to find his profile and then look through his studies and games to see the openings that he plays and his playing style. Therefore, he sought a solution which doesn’t take up as much of his time. When I spoke with him in person, he mentioned that he wanted to get better at the game and while the current chess platforms were allowing him to do that, he wanted something “more bespoke” to suit his needs as this would make it easier for him to get better at the game. He has ambitions to one day become a titled player and hopes that whilst he may have begun the game later than what is considered to be usual, it is not too late to attain a title awarded by the FIDE, the governing body of chess.

### The solution to the problem

My end user wishes for me to make a chess program for them, with some optional additional features if possible, so that he can prepare for his tournaments without having to run into any errors. He would like for me to try to build him an artificial intelligence robot that reflects the ability of a reasonably strong club level player as well as the creation of a new variant of the chess game which he left open for me to choose. A chess variant is a game that has been derived from the standard chess game that most people are familiar with. After further questioning about his interests and whether there was anything chess-related he watches, he mentioned the Big Bang Theory, which I have chosen as my basic theme for the new variant that I am going to create for him. My end user would also like for the program to be similar in performance to many of the chess sites that already exist and are very popular amongst chess players. He would like for there to be minimal lag time between him making the move and the move being played on the board. I have included a breakdown of each of the objectives that my end-user would like for me to include within the program as well as some which aren’t strictly necessary for what he wants but would be helpful if it was part of it in the coming pages. I think that the problem is suited to a computational approach because you can apply the computation techniques of abstraction, decomposition and algorithmic thinking to solve the problem. Abstraction is a process where you remove the parts of a problem you don’t need so that you can look at the more important parts of the problem. In the context of this problem, I can ignore the ELO system in chess as my end-user only wants a two-player game of chess where he can control both sides. The ELO system is the rating system in chess where you gain points based on how well you play in each time period. Each player who plays the game is assigned an ELO rating and gain rating for a win and lose rating for a loss. Decomposition is the process of breaking down the problem into smaller problems so that you can solve each smaller problem step-by-step until you have solved all of them and, thus, solved the problem. I have broken the problem down into smaller problems in one of the following pages, which will help me in the implementation phase of my project. Algorithmic thinking is the process of taking a systematic approach to solving a problem. It involves the creation of algorithms for each of the sub-problems so that once you have created a series of modules to solve each of the smaller problems obtained from the decomposition process, you will have created an entire program that solves the problem. An algorithm is a series of step-by-step instructions that are written to solve a given problem. In the context of this project, I am yet to implement the creation of these algorithms, but I have facilitated the creation of them due to the thorough decomposition of the problem I have carried out.

### 

### **Conversation**

End user email address: 004870@gsal.org.uk

Varun: “Hi Vishnu,

I am about to embark on a programming project as part of my A level Computer Science NEA. The main objective of this project is to solve a problem for someone (no matter how trivial) and so I was wondering whether there were any problems that you were coming across in your day-to-day life.

If you find a problem you wish for me to solve, please get back to me as soon as possible so that I can shortlist your project amongst the list of other project ideas I have received from other people.

Many thanks,

Varun”

Vishnu: “Hi Varun,

After much deliberation, I have found a problem for you to solve. I would like you to build a chess program for me. Provided I get chosen as your end-user for your project, I will share details as to why in due time.

Kind Regards,

Vishnu”

Varun: “Hi Vishnu,

Noted! I will try to get back to you ASAP to announce whether you have been chosen. I am still waiting on some other people to see whether they have anything. Otherwise, I will go with your project. I will set up a deadline for tonight at 8pm for any other people to hand in any problems they may wish for me to solve for them.”

Varun: “Hi Vishnu,

Just emailing you to let you know that I have chosen you as my end user. All the people I have emailed didn’t come back to me, so I had no choice but to accept your project idea. I will endeavour to solve your problem for you and make your experience over the next 8 or 9 months as smooth as possible.

On that note, if you could kindly bring to the table why you would like for me to embark on your chosen problem and the significance of it in your day-to-day life, it would be much appreciated.”

Vishnu: “That’s brilliant news! As you already know, I am a chess player who plays for Leeds Junior Chess Club. Being part of the club means that I must participate in tournaments frequently in order to continue playing there. However, a major issue for me is that I always require an Internet connection so that I can access my preparation. I am currently using a mixture of lichess and chess.com, which both have their advantages and disadvantages. Additionally, there has been a recent boom in the popularity of chess as a result of the COVID pandemic. More people were spending time at home doing nothing and so they used chess as a respite to get away from their busy and boring day-to-day lives. Whilst this has been great for chess in general and content creators who stream the game, it has meant that I frequently can’t play any games on my accounts on major chess websites due to “server issues”. For this reason, I would like for you to build me a chess program that will allow me to play the game of chess without an Internet connection and so that I can play against myself. The ex-World Champion Magnus Carlsen said “One of the best ways to improve at chess is to play with yourself at the chess board” so if you could make it so that I can control both sides, it will allow me to improve. Having said all the bad things about the current chess websites that exist, I like the smooth experience with which they operate so if you could try to incorporate this into your program, it would be much appreciated. Furthermore, I sometimes use chess variants such as Duck Chess and Chess960 to take a break from the standard variant so I will let you take all the control on this one. If possible, but not strictly necessary, I would like for you to make me a new chess variant. Again, not strictly necessary, but I would also like for you to try to make me a chess bot. They have been all the rage in the past few months, with chess.com releasing new bots every two weeks or so. It doesn’t have to be at the level of Stockfish, the best chess engine in the world, but if it reflects the ability of a club level player, it would be much appreciated. “

Varun: “Wow! That is very interesting. On the topic of AIs that play chess, I will try to do my best to build one. I will research what I can to find out about the process of building a chess AI as I currently don’t know much about the process. With regards to the chess variant, you wish for me to invent, I am trying to find some influence, which will help me to generate ideas for what I can make. Is there any TV programmes you watch that have some sort of chess aspect to it?”

Vishnu: “I vaguely remember there something being chess-related on The Big Bang Theory. I think Sheldon wanted to improve the game of chess, so he invented some new chess pieces. I can’t remember the series and episode, but I think you will be able to find it online.”

Varun: “That’s awesome! I’ve just got some ideas. I will come back to you some time soon to talk about design ideas for the program. Right now, I need to get my analysis and research sorted. Thanks for the project idea, Vishnu!”

#### Screenshots of conversation

A picture containing text, font, screenshot

Description automatically generated

I approached my end-user asking them whether they had any problem they wished for me to solve for them.

A picture containing text, receipt, screenshot, font

Description automatically generated

My end user reveals that they have a problem that they would like for me to solve for them.

A screenshot of a computer

Description automatically generated with low confidence

I sent the first email to some people and asked some other people verbally in order to collect a list of problems people wished for me to solve for them. This is so that I could choose a problem I wanted to solve and met the criteria of being at an A level standard.

A picture containing text, font, screenshot

Description automatically generated

I confirm Vishnu as my end user in this email. Of the people I asked, he was the only one who came back to me with a problem and so I accepted his.

A picture containing text, screenshot, font, document

Description automatically generated

In this email, Vishnu explains the problem and the significance of it in his day-to-day life. I also got him to explicitly state what he required me to do for him in terms of the aspects he would like included within the chess program. I use this email later to set up the list of objectives along with the questionnaire I got him to fill out.

A text on a white background

Description automatically generated with low confidence

Vishnu wanted me to create a new chess variant for him. Since the program is centred around him and what he would like included, I decided to further question him into his interests in order to generate some ideas. He initially gave me free rein over the creation of the new variant.

A picture containing text, font, receipt, screenshot

Description automatically generated

Upon research, I verified his statement and found out that one of the protagonists and main characters of the sitcom, Sheldon Cooper, had indeed created two new chess pieces. I then chose the Big Bang Theory as the main theme for the new variant Vishnu wants me to make.

A picture containing text, font, screenshot, algebra

Description automatically generated

## 

## **Research**

I began to research into this problem to help my end user out. There are popular many chess websites out there such as chess.com and lichess.org, as mentioned by my end user. However, when I did my research as to why these websites would not be optimal for using to prepare, I found that each of the websites had their own issues. For instance, the Chess.com servers have been overloaded with demand as they saw the highest peak in people signing up and playing chess on their website (over 10 million in one day!). As for lichess, they have maintenance breaks every day to tackle such problems, which often clash at peak times. As a result, my end-user was unable to access the resources he requires to prepare for their tournaments so approached me.

### **Game notation**

Within the game of chess, there are two types of commonly used notation:

PGN notation – This is a form of notation that provides data about an entire game that is being played.

FEN notation – This is a form of notation that provides data about a given position within a game with each possible chess position having its own unique FEN notation.

With this project, I will predominantly be using PGN notation as it seems the most sensible though there may be cases where FEN notation is required (such as the potential for a function that gets the FEN of a given position and then runs an evaluation of that chess position, deciding who has the better position).

### **Rules**

The first thing I did was research all the rules of chess, as I am reasonably familiar with them myself but wanted to ensure that Vishnu received as smooth an experience as he would on a regular chess website. To enable this, I watched several YouTube videos created by chess content creators, namely GothamChess and Chess.com, to get an idea of how each of the pieces move. I noted down the movements of each piece and any patterns I could find with the movements of each piece, which I have written up:

* Pawn
  + There are 8 of them and they occupy the 2nd rank (row) for the white pieces and 7th rank (row) for the black pieces.
  + They can either move one or two squares on the first move and then one move from that point onwards.
  + They capture other pieces diagonally.
  + There is a special move called “en passant”, which directly translates from French to mean “in passing”. It is a special move that happens when your own pawn is on the 5th rank (row) and your opponent moves an adjacent pawn two squares so that the pawns are facing each other laterally. For your next move only, you will be able to diagonally capture that pawn.
  + When the pawn reaches the final rank (row) possible in the direction it is facing, it can promote to anything other than a king and a pawn itself.
  + A pawn is worth one point of material.
* Knight
  + Each side has two knights which stand on opposing colours.
  + They are the only piece that can jump over your own pieces to arrive at a legal square.
  + They move in an L shape which follows the pattern one square in any direction (excluding diagonals) and then two squares perpendicular to the direction that was moved.
  + A knight is worth three points of material.
* Bishop
  + There are two of them for the black pieces and the white pieces and they stand in opposing colour complexes.
  + Unlike the knights, they stay on the same colour complex for every move.
  + There is no limit on the number of squares they can move in each turn, provided the end position falls on the same direct diagonal as the starting position.
* Rook
  + There are two of them for both sides.
  + They move up and down with no limit on the number of squares they can move but they can’t move diagonally.
  + They are involved in the special move called castling where the king move two squares. The King will move two squares and the rook will move to the left of the king.
  + Each rook is worth five points of material.
* Queen
  + The queen is the most powerful piece of all the pieces that have a material value.
  + The queen can move in any direction and follows the movements of the rook and bishop combined.
  + Each side only has one queen to start with.
  + A queen is worth 9 points of material.
* King
  + The king is most valued piece on the board as it is the piece that wins you the game of chess by checkmating the king.
  + “Checkmating the king” means attacking your opponent’s king with your pieces until the king can’t make any more moves, the direct line of attack can’t be blocked by any of your opponent’s pieces and the piece that is making the direct attack can’t be captured by one of your opponent’s pieces.
  + The king can only move in one square in any direction except in one instance called castling.
  + In castling, your king moves two places to the left or to the right and your rook will follow and end up to the left of the king.
  + When an opponent’s piece is put in the direct line of fire to your king, it is called a check and the only way to stop this is to either move your king, block the line of fire with which your opponent’s piece is attacking your king or capture the offending piece.

#### General rules

1. The white pieces move first, and the black pieces follow, and the game will continue to follow this pattern until a result can be deduced from the game.
2. If a player’s king is in check and has no more legal moves, the game ends. If the king is not in check and there are no more legal moves, a stalemate has been reached and the game ends in a draw.

### 

#### Additional rules as part of my variant Big Bang Chess

* Serpent:
  + The serpent is a chess piece which slithers around the board in a vector of (0, 3) i.e. the piece can only move forwards.
  + It poisons your opponent’s pieces
  + Each player gets two serpents but the serpents themselves are immune to each other’s poison.
* Grand Empress:
  + The old woman is a chess piece which acts as a healer.
  + Its movement is similar to that of a knight.
  + It can heal a poisoned piece provided the poisoned piece is within a 1 square radius of the old woman otherwise the poisoned piece is removed from the board.
  + It sucks up the poison and the serpent and the empress are removed from the board if the piece is nearby.

In the Big Bang Theory, Sheldon makes the rules such that the old woman turns into a Grand Empress, which is shown by her gaining a crown. She then possesses the power of the knight, queen and serpent. However, I wanted to make this variant my own, so I decided to slightly alter Sheldon’s rules. Additionally, the middle of the board is tessellated with triangles (made of triangles arranged in a repeating pattern) in Sheldon’s version but being able to implement this into a visual display and assigning some form of coordinate system would be extremely complicated so I decided against this. I chose to stick to a normal 8x8 chessboard but there is the potential to change the size of the chessboard to incorporate more pieces.

### A screenshot of a computer Description automatically generated**Proof of concept**

As part of my proof of concept, I decided to investigate the approach that has been taken by people who have made a chess program already and have published the source code online. I began to do some more research into whether it was possible and surely enough, it was. I found out that one of the major chess websites, lichess.org, was written in the object-oriented language, Scala, and so this proved that the game of chess could be developed using a programming language. Having found several resources online, I sought a solution which involved a programming language that I was familiar with, though it need not be necessary, as I felt that the development timeline would be hastened. I found many implementations of chess games in Python online and so; I decided that this project idea would be perfect for me and so I approved it as my project idea (as shown in the conversation with my end user on the previous page). Using the skills document, which has been provided by AQA, I then began to decompose the problem down into its simplest form and the most optimal approach, which demonstrate several of the skills which fall under the Group A category. (I have written this up in one of the following pages). Many of the programs I found online were making the use of the pre-built library called python-chess, which had a range of pre-programmed functions that allowed you to run the game of chess in your own terminal window. However, I decided against using that library as I felt that I would not benefit from it greatly since I chose that the majority of my project should be built from user-defined algorithms. Therefore, I plan to use the documentation of the library while implementing the implementation phase of my project to get an idea of the approach taken to make the functions within the library, should I get stuck along the development of my program. It then occurred to me that if one of the larger chess websites has used object-oriented programming to build their application, then I could perhaps take an object-oriented approach in Python to solving my end-user’s problem.

### **Idea for a new chess variant**

In the conversation, Vishnu says that he would like for me to make a new chess variant for him as he often uses the variants available online to get away from playing too much standard variant chess. Since the project is centred on his own requirements, I wanted to find out about his other interests so that the new variant would appeal to him even further. Vishnu suggested the US situational comedy, The Big Bang Theory, as a source of brainstorming potential ideas and I found that this was indeed true. One of the protagonists of the show, Sheldon Cooper acted by Jim Parsons, invents two new pieces called the grand empress/old woman and the serpent. I decided to do some further research to find out whether a Big Bang Theory version of chess had indeed been created and to my luck, there was no such project found on any websites and GitHub repositories. This resulted in me making the decision to make this as my chess variant. I also used The Big Bang Theory as my inspiration for the title of this project: “Secret Agent Prep: Chess”. In one of the episodes, three of the four protagonists play a game called “Secret Agent Laser Obstacle Chess”. However, since Vishnu would be using my program to prepare for his chess tournaments, I decided to slightly tweak the name so that it suited the purpose of the project. Additionally, since the variant is intended for entertainment purposes and nothing else, I have decided that I will not implement a win condition for the game, and it will just be a fun game to capture pieces. There will be no check, en passant, castling or checkmate.

### **Improving my knowledge foundation of Python Programming**

While I was doing my research into the proof of concept for my idea, I decided that I would improve my Python programming skills as a pivotal element of my program will be the user interface for the user to interact with the program. I discovered the PyGame module, which I could use to display images in the terminal window and create new windows altogether, which could be used to display the game itself. I then chose to do a small course in PyGame, which I felt would come in handy during the implementation phase of the project, on YouTube created by TechWithTim. The course taught me how to make a two-player space game. To increase my familiarity with the syntax, I also included comments as there were many transferrable skills which would be useful when carrying out the implementation phase of the project.

## A screenshot of a video game Description automatically generated

Red text on a white background

Description automatically generated

## A diagram with colorful text Description automatically generated with medium confidence**Decomposition of the problem**

A diagram of a board

Description automatically generated

A diagram of a project

Description automatically generated

### Use case diagram

A picture containing screenshot, text, diagram, circle

Description automatically generated

### **APIs**

API stands for application programming interface, which is a set of definitions and protocols for building and integrating application software. It is a software in-between that allows two applications to communicate. APIs are an accessible way to extract and share data within and across organisations. APIs let your product or service communicate with other products and services without having to know how they’re used.

#### APIs in the context of this project

My end user wants me to make the standard game of chess, my own variant and if possible, Vishnu also wants me to make an AI, though this isn’t a pivotal element. He wants me to make it to the standard of a club level player and the only achievable way of doing this would be through getting all of the game data from players of a similar rating and then training the AI to play moves of a similar strength. However, through some further research, I discovered that I would be requiring large volumes of data (which are readily available as shown by the pictures below), but I didn’t have the appropriate storage facility to do this, not to mention that the training process for these AI programs to play in a certain way takes a long time. Furthermore, the data doesn’t come parsed (i.e. there is no special character to separate each game’s PGN notation) so I would also have to go through the file and enter a special character into the file to separate the games. Additionally, due to the project itself having many other components and my lack of experience with programming artificial intelligence programs, I will also have to account for any learning time required to learn the necessary syntax, modules and methods I should employ to make the system.

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

Furthermore, the artificial intelligence programs that don’t require large volumes of data already have many existing versions that are already in place currently so it would be extremely difficult to find a way to make the system my own without utilising other people’s programs.

I was going to ask him whether he wanted me to implement anything else in replacement for the AI program, but he himself came up to me and asked me whether he would swap the AI for something else. I asked him what he had in mind, and he told me that he wanted me to get his online games for him so that he could analyse them. This idea seemed more approachable and doable since I found out that lichess has an API which can be utilised for developers to access account details and game data. I asked him in addition to fetching his games, if there is anything that he wants me to do for him and he said:

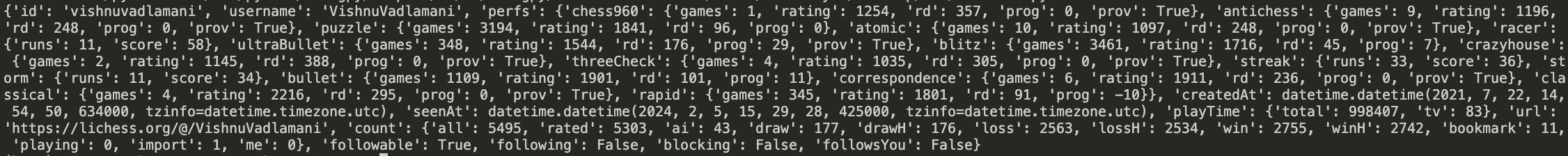
“I’m sure lichess or chess.com has many other cool features that you use in your program, so I’ll let you add something to it as well but make it linked to this somehow.”

##### Modelling the API fetch

After finding out about the API, I decided to model the API fetch as a proof of concept to see whether it would indeed be possible to execute what Vishnu has in mind. This is the code below:

A screen shot of a computer program

Description automatically generated

This is the output:

As you can see from the output above, the API data fetch has been successful and it is coming in dictionary format so I think that a JSON (JavaScript Object Notation) file will be appropriate to store the data, if only for temporary purposes. JSON is a lightweight format for storing and transporting data and so I think that it will be appropriate in this situation. Through looking at the above output, it gave me the idea to present his ratings to him graphically and upon asking him, he said:

“That would be pretty cool, but I’ll leave it up to you to decide whether it should be in there or not.”

As part of the one of the components to the project, Vishnu asked me to get his game data in a PGN format. Now, as you can see, the above code was just a trial to see whether I would be able to connect with the API (Application Programming Interface). The API also has many other functions, which can be utilised to execute what my end user is after and to solve my end-client’s requirement, I will be using the export\_by\_player() function in order to fetch the data based off some of his requirements, for example, by time format (whether the game variant is bullet, blitz, rapid or classical).

### **Parsing**

Parsing refers to the process of analysing a sequence of symbols or data to extract meaningful information or to determine its grammatical structure. In this project, I will be receiving large quantities of data from the Lichess API request and so I will need to analyse the data and pick up the parts I need before presenting it graphically. As demonstrated by the proof of concept on the previous page, you can see that in addition to the rating data, I receive a lot of other data associated with other features on lichess like Puzzle Storm and other chess variants like Atomic and Chess 960 (also referred to as Fischer’s Random) and so I will have to selectively choose the bits of data I will require from this to represent. Puzzle Storm is where you have three minutes to solve as many puzzles as you can. Fischer’s Random chess is a chess variant where all of the pieces (excluding the pawns) are shuffled in a weird order and there are 960 possible positions which can be obtained from randomly shuffling the pieces.

## **Current Systems**

There are several well-established chess websites that already exist in addition to the ones I have mentioned. In this section, I will explore the other possible solutions to my end-user’s problem and any limitations that arise from their solutions to his problem.

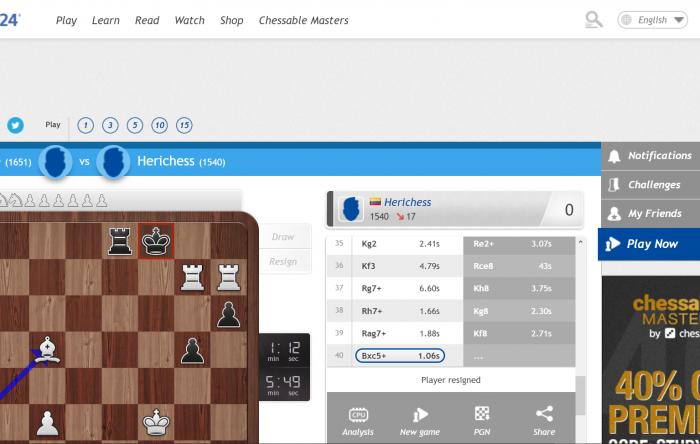
### FICS

A screenshot of a computer

Description automatically generatedThe Free Internet Chess Server (FICS) is a volunteer-run chess server that was created as a free alternative to the membership-based Internet Chess Club. It is a chess platform that was established on 3rd March 1995. It hosts over 300,000 registered users and many more regular players since they allow you to play anonymously too. This would be a good solution for my end-user as they don’t have to face the server overload issues due to several hundreds of thousands, if not millions, of people trying to access the chess.com and lichess servers at the same time. Additionally, with the large player base that they have, preparing for his tournaments by playing other people shouldn’t be too trivial, unlike some of the other existing platforms such as KasparovChess and chess24, which don’t see too many users despite several thousands of dollars being put into marketing strategies. However, there are some issues with FICS. The interface as a whole seems extremely dated. As a solution, they offer a series of interfaces which you can download to your device, but it is not strictly part of the download package when you install the FICS application. In addition to this, players tend to use a Telnet connection to interact with the server. This introduces a whole host of cybersecurity problems, which I am sure the volunteers would have sought a solution to, but it introduces a whole host of unnecessary risks. While on the Telnet connection, any data you send or request is unencrypted and so any unauthorised users, should they get access to your data, will be able to read the contents of your message. Encryption is when data is scrambled so that only the sender and receiver of the message can understand its contents. Additionally, unlike some of the other chess websites, they don’t offer any chess lessons or puzzles, which many coaches believe help you to improve at the game faster.

### chess24

Chess24 is another example of system that could be of potential use to my end-user. It is a chess website that was created by the notable grandmaster Jan Gustafsson and Enrique Guzman. They used this as a foundation to develop a plethora of chess training platforms such as Chessable where you are able to complete courses from renowned titled players such as Magnus Carlsen (ex-world champion and considered to be the greatest player of all time by many), Judit Polgar (Considered to be one of the strongest female players ever to have played the game before her retirement, which many viewed as premature at the time) and several other top players. All of these chess training websites were then merged together to form the PlayMagnus group, with the largest shareholder being Magnus Carlsen himself. They have accumulated a range of resources from lessons to puzzles to being able to find coaches to teach you as well as playing against several chess engines. As a result, it seems well-suited for my end-user to use. Furthermore, the user interface is far more up to date than FICS. However, the player base is extremely small, despite the large sums of money put into marketing strategies for promotion of their brand to increase their own brand awareness, and it takes up an extremely long time to get paired against someone. This is because the pairing algorithm works by finding players who are online, are similar in their respective ratings and are requesting to play the same time format before pairing them against each other. Due to a deficit in the number of players, it means that this time to get paired is taking unprecedented amounts of time, which makes the website sub-optimal for my end-user when it comes to preparing for their tournaments. In 2022, Chess.com made a multi-million-pound deal with the PlayMagnus Group and acquired the group itself with all of their partner companies for $83 million. From this, the active players switched from the chess platform they were on to chess.com where they were able to be paired against someone of a similar skill level with much shorter pairing times. Chess24 requires a membership to unlock all of the features needed to boost your improvement.



## **Requirements**

### Questionnaire

In order to obtain the necessary objectives for the project, I decided to send a form to Vishnu in order to get the requirements for what he needs in the chess program that I am building for him. I used the Microsoft Forms platform to create a mixture of 6 qualitative and quantitative questions that would get me the necessary data on what he needs. Additionally, should I require additional data on what he is after, I felt that I could also ask him through contacting him via email. I sent it through to him via email and I received the responses to the question extremely quickly.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer screen

Description automatically generated

Here are the responses I got from Vishnu:

A screenshot of a white and black text

Description automatically generated

I used this question to get the sort of appearance Vishnu would like for the program to have. With the existing chess websites, there are several themes that are available, and they even allow you to customise your own theme by uploading a picture from your own files. As a result, with the user in mind, I felt that this should be the first question before getting them to elaborate on what they are after. Additionally, the user interface is the first thing that they will see so I decided to target this area first before moving on.

A screenshot of a white text box

Description automatically generated

I also used this question to gauge not only the appearance of the program but also what I could incorporate from existing chess websites into my own program so that the program would suit his needs and make him feel as though he isn’t compromising on anything by using my own program rather than one already exists and has established itself to be very good.

A white background with black text

Description automatically generated

This question is the first of two to figure out whether I could incorporate something into my program that isn’t offered elsewhere and so going back to the point of him compromising on nothing, he will get additional features for free which will only allow him to attain his end goal of getting better much faster.

A screenshot of a computer

Description automatically generated

This is the second question regarding the topic area mentioned above. He mentioned buttons, which I believe is a unique feature that can’t be found on any of the major chess websites, so I am going to incorporate that into my program.

A white background with black text

Description automatically generated

This is linked to the appearance of the program and how interactive I should make it so that Vishnu will still be motivated to play chess.

A screenshot of a computer

Description automatically generated

This is also linked to the appearance as it will give me an idea for the colour scheme, I should adopt in order to make it suitable for every single time of day he is playing the game.

From the data I have received from Vishnu, I have broken the sub-problems that I have obtained from the decomposition of the problem that I have done into the necessary objectives that need to be fulfilled in order to meet my end user requirements to the standards they expect and want it to be completed to.

### Objectives

1. Create a user interface for the end-user to interact with the program.
   1. The system must be accessible and easy to use.
   2. The performance of the system must be similar to that of existing chess websites.
      1. This objective will entail trying to include minimal lag into the program itself to mimic the existing chess websites.
      2. Navigation through the program should be straight forward.
   3. Any inputs and prompts that are required from the user must be self-explanatory.
      1. There will be a sign-up button upon execution that will appear in the middle of the screen.
      2. Two fields will appear here titled “Username”, “Password”
      3. There will also be a button at the bottom of this screen titled “Login” for those people who have already made an account.
         1. They will be redirected to another page if this is pressed with two fields titled “Username” and “Password”.
         2. The user inputs to these fields will be used to check the data stored in the database to see whether there is any data matching the inputs.
   4. The interface itself will be black and everything else will be in white to contrast against the dark backdrop.
      1. My end-user wished for the appearance of the program to be “neutral”
      2. The logo of my project will appear in the top left and top right of the screen.
      3. The colour scheme will allow my end-user to be able to prepare for their tournaments regardless of time of day.
2. Creating the board
   1. After choosing one of the four options (Play against computer, 2-player chess, Big Band Chess, view previous games), a board will be created accordingly.
      1. For the “Play against the Computer” option and the “2-player chess” option, the board will consist of 64 squares in an 8x8 configuration shaped like a square.
      2. The variant I am created will use a board with a 10x10 configuration.
   2. The user will be prompted to enter two colours from a list of colours that they would like for the squares on the board to be, which will be arranged in an alternating pattern.
      1. On a standard chess board, there will be black and white colours with the queens in their own colour and the bottom right square of the board being located on the right side of the person playing with the white pieces.
   3. The user will then be prompted to select a piece set from a list of piece sets which they will not be able to change during the game.
      1. These pieces sets will come from both chess.com and lichess.org.
      2. I will also try to make my own piece set, aimed specifically for the variant I am trying to create.
3. Create chess pieces for both sides.
   1. Regardless of the button pressed, chess pieces will need to be created.
      1. This is where I will be relying on the dynamic generation of objects to create 8 pawns, 2 knights, 2 bishops, 2 rooks, 1 queen and 1 king for both the white pieces and the black pieces.
      2. With the new variant I am creating, there will be one extra pawn, a serpent and the old woman on a 10x10 board configuration.
   2. Pawn
      1. Diagonal captures with en passant function, when two pawns are adjacent other and are laterally facing each other on the fifth or fourth rank (row), which is only available for the next move.
      2. They can move one or two squares on the first move and then move one square at a time from then.
      3. When the pawn reaches the other side, it can promote to a queen, bishop, rook or knight.
   3. Knight
      1. The knight moves in an L shape with one square in any direction (excluding diagonal movements) and then squares from the new square in a perpendicular direction to the initial movement.
      2. They capture in the same way.
      3. The only piece that is able to jump over its own army.
      4. It changes the colour complex of the square it lands on every other move.
      5. Knights can move backwards.
   4. Bishop
      1. Moves diagonally on the same colour complex.
      2. It has no limits on the number of squares it can move provided the end square falls on the same diagonal as the initial starting square.
      3. They capture in the same way they move.
      4. Bishops can move backwards.
   5. Rook
      1. Rooks move up and down and left and right but can’t move diagonally.
      2. There are no limits on the number of squares they can move provided the end square falls on the same file (column) or rank (row) as the initial square.
      3. They move around the king when the king is being castled.
   6. Queen
      1. The most powerful piece that has a material value.
      2. It can move in any direction but can’t jump over any pieces unlike the knight.
   7. King
      1. The most powerful piece on a given chess board.
      2. It is this piece that decides which side has won or lost in a given game of chess.
      3. It can move one square in any direction and the only exception to that rule is in castling when the king moves two squares to the left or right.
         1. Castling can only happen when the two squares the king moves to (either the left or the right) aren’t being controlled by an enemy piece (i.e., if there is a piece covering that square, the king would be in check so that makes the move illegal)
         2. You can only castle when you haven’t moved the king or the rook that is concerned when castling.
   8. Serpent
      1. One of the new pieces I have introduced as part of my new variant.
      2. It moves three squares at a time except it can only move forward.
      3. It has a special ability which fires a poison dart at an enemy piece which is four squares away (inclusive of the serpent starting square and the attacked piece starting square).
   9. Grand Empress
      1. Another piece that I have introduced as part of my new variant.
      2. It moves exactly like the knight.
      3. It has a special healing power where it removes the poison from a piece that has been attacked by the serpent.
   10. I will prompt the user to enter a square colour and a piece set from a list of piece sets to choose from to design their board.
       1. These piece sets and square colours will come from lichess.org and chess.com.
       2. I will be designing the piece set for the variant I am creating using graphics software.
4. Making a move
   1. The user will be prompted to make a move.
      1. After the move is made, there will be a detection algorithm in place to see whether the move is legal and whether it is indeed possible (i.e., whether there is the specified piece on the board that can make it to that square).
      2. If the move is legal, the piece will move accordingly, and the opponent makes their next move until checkmate is attained where the king has no legal squares to move to.
      3. With each move that is being played, there will be some sort of audio that is played in the background to show that a move has been made.
         1. For this, I will most likely download some of the sound effects from lichess.org to implement into my program.
         2. For this to happen, this will most likely involve recursion with the base case being whether a checkmate has been reached.
      4. The move made will be pushed onto the stack.
   2. If the move is illegal, an error message will be returned to the user prompting them to play another move instead or the move will not be completed.
5. Finding out whether there is a definite result.
   1. One possibility is checkmate.
      1. This is one of the key algorithms that I will have to incorporate within the program for all three of the features which my end-user has requested.
      2. It will work by checking the presence of both kings on the board after a piece has been captured
      3. If both kings are not on the board, a checkmate has been established and the game ends.
      4. If not, a checkmate has not been established and the games goes on.
   2. Stalemate
      1. For a stalemate to be achieved, one side must have no legal moves that can be played, and their king must not be in check.
      2. This will again require the legal move finder algorithm which will find all the legal moves in a given position.
      3. If there aren’t any legal moves stored within the list, a stalemate has been established and the game ends in a draw.
   3. For the variant, there is no winning or drawing conditions. There is also no check, en passant or castling.
6. Navigation between features
   1. My end-user mentioned buttons in the questionnaire.
      1. My program is going to centre on the use of buttons to ease navigation.
      2. I am going to create four buttons after login with the four options.
      3. The “View previous games feature” will read the data from the text file and return the PGN notation of the games that have been played in the past on the program.
      4. There will also be buttons going to and fro between pages to ease navigation and there will be a quit button on the top right of the program to terminate the program.
7. Drawing the rating data graphically
   1. Fetching the rating data
      1. This program will use the Lichess.org Application Programming Interface, which will allow me to use their services and algorithms to get the data I need.
      2. I will have to ask for the API token of my end user because they are made private so I will have to detail how to get the API token and then send it to me.
      3. I will have to run the methods associated with the account data like client.account.get()
      4. Direct access to the application programming interface isn’t possible so I will have to use the Python client called berserk, which was developed by the lichess developers, to access the data.
      5. The data that arrives is received in dictionary form, which is a data structure that stores a value and then another value associated with the first value, so I will have to store the received data in a JSON file before manipulating the data.
   2. Storing and manipulating the data
      1. Upon receiving the data, I will then manipulate the data that I receive so that I get the bits of data I need and then empty the JSON data file before the next request to see the rating data.
      2. From organising the data, I will then use the dictionary methods to get the ratings I need (using key values to find the ratings).
      3. After getting the exact ratings, I will then have to store the data for each time format in its own text file.
      4. In addition to the rating, I will have to store the date that the rating was fetched so that I can
   3. Drawing the graph
      1. To draw the graph, I will then have to go through the text file that is storing the data.
      2. I will plot the time that the rating was fetched on the x axis and then the rating on the y axis.
      3. I will also have to have a different line for each variant so that there is no confusion in the graphical representation and then draw a key on the diagram so that it highlights to the user which variant each line represents.
8. Outputting the game PGN of my end user’s games
   1. I will have to meet the requirements of 7.1 here as well but I will have to use a different function to get the PGN data.
      1. Upon sending the request, I will receive all of the games played by a certain account and this can be from any player on the site, not just my end user, so I will have to parameterise the function to ensure that the data that is received is strictly relevant.
   2. Parameterising the data
      1. There are many parameters for the export games function, like start date, end date and player account, for exporting the games
      2. I need to parameterise the function by my end-user’s account, the variant he’s playing and then the number of games (which my end-user has specified as 10 at a time).
   3. Outputting the data
      1. From the data I receive from the API request, I will then have to iterate through the list of games I receive and then output each game.

## **The main algorithms**

Each aspect of the project has its own main algorithm. With the standard chess game and the Big Bang Chess variant, it is going to be shared between several different ones which are:

* Fetch move – This algorithm, as it suggests, is going to get the move from the user. This will work by fetching every legal move that can be played at a given turn and if the move that is inputted is legal, the move will be returned and then it will be played. If the move is not legal, the move will not be played.
* Checkmate checker or checking whether a king is on the board – Throughout the game, for every move that is being played, there will have to be an algorithm which checks whether a checkmate has occurred before asking the user to input the next move they wish to be played. This algorithm will check the legal moves that can be played by the king. If there are any, it will ask for the user’s next move and the game continues. If there aren’t any, a checkmate has been established and the game ends. Alternatively, the algorithm could also be to test whether the king is captured

if click\_coords in black\_locations: # if the second set of coordinates is in the black\_locations (i.e. where the black pieces are)

black\_piece =black\_locations.index(click\_coords) # get the index of the piece that is being attacked and clicked on

if black\_pieces[black\_piece] == 'king': # If the piece is the king,

winner = 'white' # White wins.

black\_pieces.pop(black\_piece) # remove the piece from the board

black\_locations.pop(black\_piece) # remove the piece from the list of black locations

black\_moved.pop(black\_piece) # remove the piece from the list of pieces that can be moved.

* Display board – This is the algorithm which will not only create the board but will also make any changed to the board that have occurred as a result of the user making moves.

*def* draw\_board(*hex\_colour1*, *hex\_colour2*):

# This function is responsible for drawing the board out.

#The two parameters that are inputted into the function are the two colours in hex form which are selected by the user from the getSquareColour() function.

row\_count, rank\_count = 8, 8 # The number of rows and ranks on a chess board

cell\_size = 100

board\_colour1 = pygame.Color(*hex\_colour1*) #Initialise the first colour selected by the user.

board\_colour2 = pygame.Color(*hex\_colour2*) # initialise the second colour selected by the user

border\_colour = pygame.Color('gold') # Border colour

# Precompute rectangles for the entire board

board\_rects = [[pygame.Rect(col \* cell\_size, row \* cell\_size, cell\_size, cell\_size) for col in range(rank\_count)] for row in range(row\_count)] # Initialising a rectangle for each square.

#Through using the rectangle to represent a square, we can in turn avoid creating a separate Square class.

# Draw the entire board with alternating colours using a single draw.rect call

pygame.draw.rect(screen, border\_colour, pygame.Rect(0, 0, cell\_size \* rank\_count, cell\_size \* row\_count), 5)

for row in range(row\_count):

for col in range(rank\_count):

if (row + col) % 2 == 0: # A simple algorithm which has binary results and so this can be used to draw the alternating colour pattern on the squares.

colour = board\_colour1

else:

colour = board\_colour2

pygame.draw.rect(screen, colour, board\_rects[row][col])

# Draw the "FORFEIT" text

screen.blit(medium\_font.render('Resign', True, 'black'), (810, 830)) # Resign button

if white\_promote or black\_promote:

pygame.draw.rect(screen, 'gray', [0, 800, WIDTH - 200, 100])

pygame.draw.rect(screen, 'gold', [0, 800, WIDTH - 200, 100], 5)

screen.blit(big\_font.render('Select Piece to Promote Pawn', True, 'black'), (20, 820))

With the view the games feature, the main algorithm will be:

Fetch game data – This algorithm will read the text file that has stored the games played by a given user and return the game data in the form of PGN notation.

A screen shot of a computer program

Description automatically generated

# **Design**

## User interface – how it looks to the user

For my project which my client has asked me to complete, I first thought that I should decide upon the module I wanted to use for it. Due to familiarity with PyGame after the completion of the online course, I decided that I would try to stick with that for my user interface (UI). As a backup plan, my idea is to use tkinter as it is something which I am relatively familiar with through coding competitions in school. I decided to start with creating designs for my user interface first so that when the summer holidays started, I would have a rough idea of what my client wants the system to look like before I started implementing everything he wanted. This is so that after making my system, I wouldn’t have to make any major changes to my user interface. My user asked for neutral colours as part of the initial discussions and so I decided to ask him about whether he really wanted neutral colours or whether he would be open to having some colours.

Upon asking him, he said:  
  
“I thought that neutral colours would look really good, but now that I think about it, I don’t think it would look that good. Yeah, I don’t mind if you include some colours on the game but try to stick to mostly neutral schemes”.

A screenshot of a computer

Description automatically generatedFrom this conversation I had with him in person, this led me to make my first design decision with regards to the user interface. My client didn’t specify any specific colours he wanted me to include so therefore I’ve decided that I will stick to his brief as much as possible. From modelling some of the requirements he wanted me to meet, I created the following user interface with PyGame:

Upon showing this to my end-user, he told me that he liked the colour scheme but to swap the dark green for another colour. I responded by telling him that this was just a model, and that the real system would incorporate better colours and have much more time and thought put into deciding on the colours.

After deciding on some of the colours that my end-user wanted in the user interface, I noticed that my client liked a lot of the features associated with chess.com. This influenced my decision on designing the application and I thought that I should make a Create Account and Log In system for it as I thought it would appeal to my friend.

From the requirements for the user interface, I was able to fetch from my end-user from the above design, I have now designed the user interface on the next page, which my end-user is happy with. I also chose to use this design for the Log in Screen since my end user was happy with this but chose not to model it since I already had a rough idea of what I wanted to achieve from modelling my user interface.

Since my end user changed his opinion about the colour scheme of the user interface, I chose to make the colour schemes for the games components themselves more customisable, similar to that of the currently existing systems, so that he could choose the colour scheme according to what he wants rather than enforcing a particular colour scheme (though “neutral” colour scheme was mentioned in the questionnaire which reflects in the requirements I have set to achieve).

A chess board with chess pieces

Description automatically generatedA chess board with chess pieces

Description automatically generatedA chess board with chess pieces

Description automatically generatedA game of chess with different objects

Description automatically generated**A screenshot of a computer

Description automatically generatedPiece sets and colours available**

Anarcandy Basic Merida 3D

Square colours: cream, orange, yellow, green, light green, dark green, light blue, pink, purple, brown, grey, white.

Background colours: red, orange, yellow, green, light green, dark green, blue, light blue, dark blue, pink, purple, brown, grey, white.

## The games components

With the game itself, I did a lot of research into how I could possibly implement the game. Before doing any of this, I initially thought about using lists to represent the rows, but then found the problem that it would be much harder to map the changes of positions across different lists for the different pieces, not to mention the additional rules that will also be present as part of my variant. I then saw some solutions developed by other users which used will follow a tutorial from codewithfaraz.com to develop the main part of the standard chess game. This is because there are so many different implementations of the game that I found it difficult to find a way to create my own. However, I will change the implementations of the algorithms themselves because the coding style employed on that website is extremely poor. There are lots of inefficiencies within the subroutines that control the piece movement and other parts of the game logic, as well as a lack of loose coupling and the use of global variables, which I am going to eliminate. I will also add my own functionalities into the game so that I make the game my own rather than having followed a tutorial.

A screenshot of a black screen

Description automatically generatedIn order to make the game with a graphical user interface, I noticed that the only possible way to go about this was to create an array implementation or a matrix implementation. For the standard game of chess and the Big Bang Variant, the matrix implementation will be used. However, upon first viewing for the standard game of chess, I noticed that the code was extremely unstructured and so when it comes to making the Big Bang Variant, I will change the implementation so that it incorporates classes and objects and has a better structure than the purely procedural and modular approach that was assumed by the creator of the codewithfaraz.com website.

For the View Lichess Stats Feature, I will be using the matplotlib library (which I am familiar with through independent projects in my own time) to draw the graph of ratings and the rest of the code will run in the terminal.

### Code structure employed for each component

For the standard variant, I will stick with the procedural programming method that has been employed by the website but change the algorithms themselves to encourage loose coupling. For the Big Bang Variant, I will utilise object-oriented programming techniques to deliver a better and more efficient solution. I will create classes to represent each of the different pieces before implementing my own functionalities for the new pieces I am going to create: the serpent and the grand empress. For the final component, the View Lichess Stats feature, I have implemented a procedural approach with loose coupling. I will implement a different file for each class and each data file will have its own name as well.

## **Structure diagram for the games components**

### Big Bang Variant

Location

Square

Pawn

Piece

Serpent

Rook

Black pieces

White pieces

Player 2

Player 1

King

Grand Empress

Queen

Bishop

Board

Objects

Entities

Big Bang Variant

### Standard Variant

There will be no objects in this component of the game, unlike in the Big Bang Variant, as I have decided to implement this using procedural programming, as shown by the tutorial on the codewithfaraz.com website (referenced in the bibliography). This is in order to demonstrate more coding styles within my implementation when the time arises that I have to build my program.

Standard Variant

Black pieces

White pieces

Player 2

Player 1

Entities

Location

Square

Rook

Queen

Pawn

Knight

Bishop

King

Pieces

Board

## **Data Flow Diagram**

The user enters a username and password on the create account screen to make an account and enters the same credentials, provided they exist in the database, on the login screen as well.

LoginDetails.db

Submit Details

Account Details

**Login system**

Check details presence in the database

Account Details

## **A diagram of a process Description automatically generatedSystem Flowchart**

The system flowchart on the previous page shows the rough structure I am aiming to implement for the majority of the project. It shows the breakdown for the standard chess game and the Big Bang Variant but due to me having to utilise multiple graphics modules to complete these components, I anticipate that there will be problems with integrating the View Lichess Stats feature onto the main application, which is why I will ask the user to run the feature on a different file instead.

### Standard Chess Variant Flowchart

A diagram of a computer

Description automatically generated

The program will first fetch the background colour, the two square colours and the piece set, which is part of the customisable features. It will then draw the board and then draw the pieces. The first check will be the checkmate check to see if the kings are still on the board. If the kings are still on the board, it will then ask the user to input a move. If the move is legal, it will then play the move and then switch the turn to the black pieces. If the move is not legal, it will not play the move and the same side will have to play another move. This loop will then continue during the whole game until a result has been established, which is done using the checkmate checker algorithm as well as looking at the legal moves algorithms.

### Big Bang Variant Flowchart

A diagram of a computer

Description automatically generated

The Big Bang Variant will carry an even simpler implementation. It will check whether a move is legal and then play it and since it was designed for entertainment purposes only, there will be no checkmate either.

### View Lichess Stats Flowchart

A diagram of a program

Description automatically generated

The View Lichess Stats Component will work by getting the user choice from 1,2 or quit. If the input is 1, it will get the game PGN based off the variant that is inputted from the user. If the input is 2, it will graphically represent the ratings by variant. If ‘quit’ has been inputted, the system will exit.

## **UML Diagrams**

### Big Bang Variant

As I’ve already mentioned, the tutorial I will follow for the standard game of chess doesn’t involve or use any classes whatsoever so in order for better programming and style, I am going to implement my Big Bang Variant using classes. Below is the UML diagram for the standard variant:

A diagram of a class

Description automatically generated

Each of these classes will be in their own separate files to ensure organised file structure and it will mean that I will be able to access the classes I need more easily, not to mention making it easier to find something within the program. The attributes and methods in this class will all be protected (functionalities and stats associated with that object can only be used in those classes and sub classes related to the object itself) rather than public (the functionalities and attributes can be used anywhere in the code) and private (the functionalities and attributes can only be used within the class that they are defined) because upon instantiation (creating these objects), I will be utilising their \_value attributes to work the total value of the material that has been captured in another class outside of the class that the attribute is associated with. I will then display the total material that has been captured by the user once the game is over. The Big Bang Variant doesn’t actually have a winning condition because my end user wanted it purely for entertainment purposes and so I made it so that there are new and unique piece functionalities, and you just capture the pieces on the board. There is also no en passant, castling or check, so you can capture the king prematurely and it will not end the game.

### Running the Big Bang Variant Game

There are two classes associated with running the Big Bang Variant Game (there is composition aggregation):

A screenshot of a computer

Description automatically generated

I will implement the Big Bang App as an object-oriented program with private and protected variables and methods.

### UML for my user interface

**A computer screen shot of a diagram

Description automatically generated**

For each of the screens in my user interface, I will make them into objects so that it is easier to call and switch between frames. In addition to this, I will also implement a stack data structure to enable the functionality of going back and forth between screens. When moving between screens, the stack will store the screen that was displayed before the one that is currently being displayed. If a Back button is pressed, the stack will utilise the pop function to return the screen at the top of the stack (i.e. the screen object that was last added to the stack).

## **Database design**

Before trialling storing the data with databases, I initially thought that it would be better to store the data using text files as it would be easier to access and faster processing. However, I quickly realised that there would be a lack of security and so I planned to remove the password field in the login system but then realised that there would then be no point in having it there. Therefore, I decided to use a database as it incorporated both the features of storing the data itself and storing the data securely, preventing any users from accessing the data.

As part of the login system, I am storing the login data in a database. I first decided on the fields that I will need to store in the table as part of the login process and from this, I saw that the best field names to use to design the database are:

* userID
* username
* passwordID
* plaintext password
* hashed password

Following this, I decided to build the tables in the most optimal way in third normal form. This entailed developing possible permutations based on my requirements and then breaking the tables down further through following the normalisation principles. In order to ensure that a table is in 1NF,

### The tables

#### PasswordDetails Table

A close up of a computer screen

Description automatically generated

Primary key: passwordID

#### IDDetails Table



Primary key: userID

Foreign key: passwordID references PasswordDetails(passwordID)

#### UsernameDetails Table

A screenshot of a computer

Description automatically generated

Primary key: userID

Unique constraint: username

Foreign key: userID references IDDetails(userID)

### **E-R diagrams for the above system**

UsernameDetails

PasswordDetails

IDDetails

To enable this authentication system and storing these details on the database, I looked into some modules that I could implement this with. Before doing any research into this, I thought that I could potentially use Microsoft Access as a medium to hold the login details in the database before establishing some sort of connection to this database via a module. However, I came across two modules that would enable me to do just this and far simpler: mySQL and sqlite3. I looked through the documentations of the two modules and felt that sqlite3 would be far more appropriate due to the small scale of users that will be using the system. Through using the w3schools website, I was able to pick up some of the key syntax statements and concepts, which allowed me to gain the skills required to make a login system with a database backend to store the data. After this, I modelled the potential SQL queries that I will be required to make as shown below:

Upon creating a new account:

A screen shot of a computer program

Description automatically generated

This query checks whether the username that is entered by the user exists in the Users table. If it does, an error is returned informing them that the username already exists and then directing them to the LogIn screen to login.

Upon logging in with the creating credentials, these are the steps that will be undertaken in the check\_login algorithm:

* Check whether the username exists in the database
* Check whether the password entered by the user matches the corresponding password field for the same entry in the username field.
* Check whether the hash value produced from hashing the user’s password matches the corresponding hash value for the same record in the database.
* If it does, transfer the user to the Main Menu screen as they have been authenticated.
* If it doesn’t meet these three checks, output an error saying invalid credentials.

### The Data Definition Language breakdown of the database

CREATE DATABASE LoginDetails.db

CREATE TABLE IF NOT EXISTS PasswordDetails (

passwordID INTEGER PRIMARY KEY AUTOINCREMENT,

hashed\_password TEXT

);

CREATE TABLE IF NOT EXISTS IDDetails (

userID INTEGER PRIMARY KEY AUTOINCREMENT,

passwordID INTEGER,

FOREIGN KEY(passwordID) REFERENCES PasswordDetails(passwordID)

);

CREATE TABLE IF NOT EXISTS UsernameDetails (

userID INTEGER PRIMARY KEY,

username TEXT UNIQUE,

FOREIGN KEY(userID) REFERENCES IDDetails(userID)

);

## **Key algorithms**

### Checkmate checker pseudocode

Due to the complex functionality of having to find the moves that block the check, I have instead made the design decision to flash the king in check to the user and the game will end when a king is captured. This will be brought in from the video tutorial I will be using, and I will also add the extra functionality of keeping a material count advantage as in the currently existing systems, they all store the list of pieces that have been captured and the value of these pieces summed together.

DEF SUBROUTINE CHECKMATE\_CHECKER(legal\_moves):

# This function shows the abstracted pseudocode function for checking whether a checkmate has occurred. I have also added the stalemate check as well but in the actual game, this will implemented separately to encourage loose coupling.

If it’s white’s turn and the king has been captured:

Result = “black wins!”

Else if it’s black’s turn and the king has been captured:

Result = “white wins!”

Else if it’s white’s turn or if it’s black turn and there are no legal moves:

Result = “Stalemate!”

### Checkmate checker flowchart

This flowchart shows an algorithm that will be utilised in only the Standard Variant as there is no need nor requirement to check for checkmate in the Big Bang Variant.

A diagram of a game

Description automatically generated

There are three possible end conditions: win, lose or draw. So, in each eventuality, white can win, black can win, or it’s a draw. Therefore, since my checkmate algorithm is checking for the presence of the king on the board, I simply check in each of the piece lists as to whether the king piece is present in them. If both kings are present, the game still goes on. If one king isn’t on the board, we determine who is the winner based on which side’s turn it currently is. The draw algorithm is independent of this as to check for a draw, it depends on getting the list of legal moves that can be played.

### Draw board algorithmA computer code on a blue background Description automatically generatedA screen shot of a computer program Description automatically generated

The chess board forms an alternating pattern so I can iterate from 0 to 64 (which represents how many squares are on the board) and then using the remainder divison operation (so that when I divide by two, it either returns a 1 or a 0), I can draw the corresponding square colour, accordingly, forming an alternating pattern.

### Fetch move algorithm pseudocode

This abstracted pseudocode function is responsible for fetching the move from the user to be played on the board.

Def SUBROUTINE FetchMove(move, legal\_moves):

# For both the black pieces and the white pieces

If a piece is clicked:

If that move that the user has inputted is legal:

Return that move

Else:

Return that move and wait for a valid move input.

Else:

Just wait for the input.

#### **Fetch Move algorithm flowchart**

This will be used in the Big Bang Variant and the Standard Chess Variant albeit they will be altered slightly to better suit their respective circumstances.

A diagram of a computer

Description automatically generated

The fetch move algorithm works after the legal moves of a given position have been fetched from the get legal moves function. The user will select a piece and then select a location to move the piece to. If the move is legal, the move will be returned and then the turn will be switched to the opponent’s side. After the board has been updated, then function will be executed again to run for the opponent’s turn. If the move inputted is not legal, it will go back to the starting state of getting the move input from the user. If no move has been inputted, nothing will happen, and the system will run a loop where it is waiting for the user to enter a move.

## **Data structures**

### Stack

If my user interface is using PyGame for the entire system, I will not be using any data structures. However, if I am using tkinter for my user interface, I will ensure that each frame is an object and so when it comes to navigating between screens, I will use a stack to store the previous screen objects and if the back button is pressed at any time, it will use the pop function in the stack to retrieve the previous screen that has been accessed.

The stack code:

**class** StackException(Exception):

**def** \_\_init\_\_(self, value):

self.value = value

**def** \_\_str\_\_(self):

**return** self.value

**class** Stack(object):

**def** \_\_init\_\_(self, size):

self.\_\_TOS = 0

self.\_\_size = size

self.\_\_array = []

**def** isEmpty(self):

**return** self.\_\_TOS == 0

**def** isFull(self):

**return** self.\_\_TOS == self.\_\_size

**def** peek(self):

**if** self.isEmpty():

**raise** StackException("Stack is empty")

**else**:

**return** self.\_\_array[self.\_\_TOS - 1]

**def** push(self, item):

**if** self.isFull():

# messagebox.askyesno(message="You seem to just be going in between screens! Taking you back to the homepage...")

**raise** StackException("Stack is full")

**else**:

self.\_\_array.append(item)

self.\_\_TOS += 1

**def** pop(self):

**if** self.isEmpty():

**raise** StackException("Stack is empty")

**else**:

self.\_\_TOS -= 1

**return** self.\_\_array.pop()

**def** display(self):

**print**("Screen History Stack:")

**for** screen\_class **in** reversed(self.\_\_array[:self.\_\_TOS]):

**print**(screen\_class.\_\_name\_\_)

**def** clear(self):

self.\_\_array = []

self.\_\_TOS = 0

I will be using the Stack Object to store previously accessed screens and for every time that the back button is pressed, it will pop the last screen used from the stack and then display it.

# **Implementation**

## **Calling parameterised Web service APIs and parsing JSON to service a complex client server model**

[**CallingWebAPIsExample1**](#CallingWebAPIsExample1) Page 84, return\_lichess\_data(), View Lichess Stats feature

[CallingWebAPIsExample2](#CallingWebAPIsExample2) page 84, fetch\_games() , View Lichess Stats feature

[ParsingJSONExample1](#ParsingJSONExample1) Page 84, return\_lichess\_data(), View Lichess Stats feature

[ParsingJSONExample2](#ParsingJSONExample2) Page 86, get\_rating\_data(), View Lichess Stats feature

[ParsingJSONExample3](#ParsingJSONExample3) Page 86, get\_rating\_data, View Lichess Stats Feature

## **Cross table SQL**

### **SQL tables**

[SQLTable1](#SQLTable1) – Page 67, \_\_submit\_details(), used as part of my create account system to store details

[SQLTable2](#SQLTable2) – Page 67, \_\_submit\_details(), used as part of my create account system to store details

[SQLTable3](#SQLTable3) – Page 68, \_\_submit\_details(), used as part of my create account system to store details

### **SQL functions**

[SQLFunction1](#SQLFunction1) – page 68, \_\_submit\_details(), used as part of my create account system

[SQLFunction2](#SQLFunction2) – Page 68, \_\_submit\_details(), used as part of my create account system

[**SQLFunction3**](#SQLFunction3) – page 68, \_\_submit\_details(), used as part of my create account system

[SQLFunction4](#SQLFunction4) – page 68, \_\_submit\_details(), used as part of my create account system

## **List operations**

[ListOperationsAppendExample1](#ListOperationsAppendExample1) – page 87, Used to store the rating from the View Lichess Stats feature in a list temporarily.

[ListOperationsAppendExample2](#ListOperationsAppendExample2) – page 91, adding items to a list if there is one of your own pieces nearby in the Big Bang Variant.

[ListOperationsCopyExample1](#ListOperationsCopyExample1) – page 55 – creating a copy of the new castling moves list in the normal game.

[ListOperationsPopExample1](#ListOperationsPopExample1) – page 80 – removing a piece from the board in the normal game + variant.

[ListOperationsPopExample2](#ListOperationsPopExample2) – page 81 - removing a piece from the board in the normal game + variant.

## **Recursive algorithms**

[RecursiveAlgorithmsExample](#RecursiveAlgorithmsExample) – page 51 – Used to fetch two colours for the squares.

## **Hashing**

[HashingAlgorithmExample1](#HashingAlgorithmExample1) – page 69 – the hashing function itself.

[HashingAlgorithmExample2](#HashingAlgorithmExample2) – page 70 – the same hashing function

[HashingUsageExample1](#HashingUsageExample1) – page 68- Hash the password in the CreateAccountScreen

[HashingUsageExample2](#HashingUsageExample2) – page 71 – Used in LoginScreen to compare hash values in the database for the hash value calculated from the user entering the details and the password being hashed to produce a new value.

## **Stack/Queue operations**

[StackClass](#StackClass) – page 110

[StackClassUsage](#StackClassUsage) – page 74

[StackOperationsExample1](#StackOperationsExample1) – page 74

[StackOperationsExample2](#StackOperationsExample2) – page 74

## **Dynamic generation of objects based on complex user-defined use of OOP model**

### Inheritance

[OOPModelExample4](#OOPModelExample4) – page 104, Serpent(), used as part of the Big Bang Variant

[OOPModelExample5](#OOPModelExample5) – page 105, GrandEmpress(), used as part of the Big Bang Variant

[OOPModelExample6](#OOPModelExample6) – page 106, King(), used as part of the Big Bang Variant

[OOPModelExample7](#OOPModelExample7) – page 107, Queen(), used as part of the Big Bang Variant

[OOPModelExample8](#OOPModelExample8) – page 107, Rook(), used as part of the Big Bang Variant

[OOPModelExample9](#OOPModelExample9) – page 108, Bishop(), used as part of the Big Bang Variant

[OOPModelExample10](#OOPModelExample10) – page 109, Pawn(), used as part of the Big Bang Variant

[OOPModelExample12](#OOPModelExample12) – page 65, CreateAccountScreen(), used as part of my user interface and the create account system

[OOPModelExample13](#OOPModelExample13) – page 69, LogInScreen(), used as part of my user interface and the login system

[OOPModelExample14](#OOPModelExample14) – page 72, MainMenuScreen(), used as part of my user interface and the options system

[OOPModelExample15](#OOPModelExample15) – page 73, App(), used to setup the user interface

### Polymorphism

[OOPModelExample4](#OOPModelExample4) – page 104, Serpent(), used as part of the Big Bang Variant

[OOPModelExample5](#OOPModelExample5) – page 105, GrandEmpress(), used as part of the Big Bang Variant

[OOPModelExample6](#OOPModelExample6) – page 106, King(), used as part of the Big Bang Variant

[OOPModelExample7](#OOPModelExample7) – page 107, Queen(), used as part of the Big Bang Variant

[OOPModelExample8](#OOPModelExample8) – page 107, Rook(), used as part of the Big Bang Variant

[OOPModelExample9](#OOPModelExample9) – page 108, Bishop(), used as part of the Big Bang Variant

[OOPModelExample10](#OOPModelExample10) – page 109, Pawn(), used as part of the Big Bang Variant

[OOPModelExample13](#OOPModelExample13) – page 69, LogInScreen(), used as part of my user interface and the login system

### Interfaces

[OOPModelExample1](#OOPModelExample1) – page 88, BBApp(), App class which is used to run the Big Bang Variant.

[OOPModelExample2](#OOPModelExample2) – page 97, Board(), Class with all of the Big Bang Variant functions

[OOPModelExample12](#OOPModelExample12) – page 65, CreateAccountScreen(), used as part of my user interface and the create account system

[OOPModelExample16](#OOPModelExample16) – page 93, this is an example of encapsulation through the use of private variables

[OOPModelExample17](#OOPModelExample17) – page 96, this is an example of encapsulation through the use of private variables

### Composition

[OOPModelExample1](#OOPModelExample1) – page 88, BBApp(), composition aggregation with the Board() class, used in the variant

### Classes

[OOPModelExample1](#OOPModelExample1) – page 88, BBApp(), used as part of the Big Bang Variant to execute the game.

[OOPModelExample2](#OOPModelExample2) – page 97, Board(), used as part of the Big Bang Variant to store all of the methods associated with the game.

[OOPModelExample11](#OOPModelExample11) – page 110, Stack(), used as part of the user interface to go back to previous screens

[OOPModelExample19](#OOPModelExample19) – page 101, used to call the legal move fetcher for every piece class

[OOPModelExample20](#OOPModelExample20) – page 101, used to call the legal move fetcher for every piece class

[OOPModelExample21](#OOPModelExample21) – page 107, used to get half of the queen’s functionality (the other half comes from the bishop).

[OOPModelExample22](#OOPModelExample22) – page 93

[OOPModelExample18](#OOPModelExample18) – page 96

[ExceptionClass1](#ExceptionClass1) - page 52 – ColourError() - Used in Big Bang Variant and the Standard Variant to handle any exceptions from selecting colours that aren’t allowed.

[ExceptionClass2](#ExceptionClass2) – StackException() – page 110 – Used if the top of stack is 0 and you try to pop the previously accessed screen from the class and if the stack is full and you try to access the next screen.

## **Writing and reading from text files**

[TextFilesUsageExample1](#TextFilesUsageExample1) – page 111, View Lichess Stats Feature: ratings graph, used to store the ratings of my end user so that it can be drawn on a graph

[TextFilesUsageExample2](#TextFilesUsageExample2) – page 111, View Lichess Stats feature: ratings graph, used to store the ratings of my end user so that it can be drawn on a graph

[TextFilesUsageExample3](#TextFilesUsageExample3) – View Lichess stats feature: ratings graph, page 112, used to store the ratings of my end user so that it can be drawn on a graph

[TextFilesUsageExample4](#TextFilesUsageExample4) – page 112, View Lichess stats feature: ratings graph used to store the ratings of my end user so that it can be drawn on a graph

## **Dictionaries**

[DictionariesExample1](#DictionariesExample1) – page 53, used to get the colour requirements for the user interface for the Big Bang Variant and the normal chess game.

[DictionariesExample2](#DictionariesExample2) – page 65, used to get the piece set requirements for the user interface for the Big Bang Variant and the normal chess game.

[DictionariesExample3](#DictionariesExample3) – page 74, used to initialise and scale image sizes as part of the Big Bang Variant and the Standard Variant.

[DictionariesExample4](#DictionariesExample4) – page 86, to temporarily store the dates and ratings of the end user as part of the View Lichess Stats feature: ratings graph.

## **Complex Mathematical model**

[Matrix1](#Matrix1) – Page 98, a matrix is used to represent the coordinate system on the board in the Standard Variant and the Big Bang Variant.

[Matrix2](#Matrix2) – Page 91, used in the Big Bang Variant to look at the neighbouring squares to the specific position, which is represented as a matrix.

[MathsModel1](#MathsModel1) – Page 93, after a piece has been captured, the value of that piece is added to the material count variable. It is a simple addition functionality.

## **Single Dimension Array**

[SingleDimensionArrays](#SingleDimensionArrays) – Page 98, used in the Big Bang Variant and the Standard Variant (on a different page) to establish necessary game variables such as storing the white and black piece locations in separate arrays.

## **The code**

### **home.py**

**import** tkinter **as** tk

**from** tkinter **import** messagebox

**from** PIL **import** Image, ImageTk

**import** warnings

**import** pygame

**import** time

**from** bigbangchessapp **import** \*

**import** sqlite3

**from** stack **import** \*

**class** ColourError(Exception): *#Exception class to catch any exceptions that happen when the background colour is being selected.*

**def** \_\_init\_\_(self, value):

self.value = value

**def** \_\_str\_\_(self):

**return** self.value

*############################################################### - Two player chess game - #########################################################*

**def** draw\_board(hex\_colour1, hex\_colour2):

*# This function is responsible for drawing the board out.*

*#The two parameters that are inputted into the function are the two colours in hex form which are selected by the user from the getSquareColour() function.*

row\_count, rank\_count = 8, 8 *# The number of rows and ranks on a chess board*

cell\_size = 100

board\_colour1 = pygame.Color(hex\_colour1) *#Initialise the first colour selected by the user.*

board\_colour2 = pygame.Color(hex\_colour2) *# initialise the second colour selected by the user*

border\_colour = pygame.Color('gold') *# Border colour*

*# Precompute rectangles for the entire board*

board\_rects = [[pygame.Rect(col \* cell\_size, row \* cell\_size, cell\_size, cell\_size) **for** col **in** range(rank\_count)] **for** row **in** range(row\_count)] *# Initialising a rectangle for each square.*

*#Through using the rectangle to represent a square, we can in turn avoid creating a separate Square class.*

*# Draw the entire board with alternating colours using a single draw.rect call*

pygame.draw.rect(screen, border\_colour, pygame.Rect(0, 0, cell\_size \* rank\_count, cell\_size \* row\_count), 5)

**for** row **in** range(row\_count):

**for** col **in** range(rank\_count):

**if** (row + col) % 2 == 0: *# A simple algorithm which has binary results and so this can be used to draw the alternating colour pattern on the squares.*

colour = board\_colour1

**else**:

colour = board\_colour2

pygame.draw.rect(screen, colour, board\_rects[row][col])

*# Draw the "FORFEIT" text*

screen.blit(medium\_font.render('Resign', True, 'black'), (810, 830)) *# Resign button*

**if** white\_promote **or** black\_promote:

pygame.draw.rect(screen, 'gray', [0, 800, WIDTH - 200, 100])

pygame.draw.rect(screen, 'gold', [0, 800, WIDTH - 200, 100], 5)

screen.blit(big\_font.render('Select Piece to Promote Pawn', True, 'black'), (20, 820))

**def** getSquareColour(colours\_picked=None, count=0):

*#This is a customisable feature. This function gets two square colours of the user's choice and then returns them as a list.*

**if** colours\_picked **is** None:

colours\_picked = []

colour\_dict = {

"cream": "#FFFACD",

"orange": "#FFA500", *# The square colours are all stored in hex form so that they can be directly utilised to initialise the colours in the draw\_board function.*

"yellow": "#FFFF00", *# The key value for this dictionary is the colour that the user will input and the pair value is the corresponding hex code for that colour.*

"green": "#008000",

"light green": "#90EE90",

"dark green": "#006400",

"light blue": "#ADD8E6",

"pink": "#FFC0CB",

"purple": "#800080",

"brown": "#A52A2A",

"grey": "#808080",

"white": "#FFFFFF"

}

**if** count < 2: *# The running condition that stops the function running more than twice.*

valid = False

**if** count == 0:

**print**("Please select a colour for the squares.**\n**")

**else**:

**print**("Please select a second colour for the squares")

**while** **not** valid:

sqcol = input(">").lower()

**if** sqcol **in** colour\_dict **and** sqcol **not** **in** colours\_picked: *# two checks: the colour is present in the dictionary and that the colour hasn't already been picked.*

valid = True

colours\_picked.append(sqcol)

**elif** sqcol **in** colours\_picked:

**print**("You have already selected that colour. Please choose a different one.")

**elif** sqcol == "black":

**print**("You can't select that colour because you won't be able to see the black pieces on the board.")

**else**:

**print**("That colour is not available. Please choose from the available colours.")

**return** getSquareColour(colours\_picked, count + 1)

**else**: *# base case - if the number of times that the function has been executed is greater than or equal to 3. After the function has been executed twice, we have satisfied the stopping condition*

hex\_codes = [colour\_dict[colour] **for** colour **in** colours\_picked] *# The hex codes which are picked up are stored in a list and then returned.*

**return** hex\_codes

**def** work\_out\_offset(pieceset):

*# When drawing the pieces on the board, the images are coming in different sizes so when drawing them, we need to offset them so that they fit on each square.*

*# This offsetting process is done for every pieceset.*

**if** pieceset == "basic": *# When each pieceset is selected, the offsetting will ensure that the images are positioned as closely to the centre as possible.*

offset\_x, offset\_y = 5, 7

**elif** pieceset == "3D":

offset\_x, offset\_y = 0, -8

**elif** pieceset == "anarcandy":

offset\_x, offset\_y = 15, 8

**elif** pieceset == "merida":

offset\_x, offset\_y = 3, 3

**return** offset\_x, offset\_y

**def** draw\_pieces(pieceset):

*#This function is responsible for drawing the pieces on the board.*

offset\_x, offset\_y = 0, 0

offset\_x, offset\_y = work\_out\_offset(pieceset) *# Get the offset values for the pieceset that has been selected.*

**for** i, (pieces, locations, colour) **in** enumerate([(white\_pieces, white\_locations, 'red'), (black\_pieces, black\_locations, 'blue')]):

**for** j, piece **in** enumerate(pieces):

index = piece\_list.index(piece)

pawn\_offset\_x, pawn\_offset\_y = 10, 0 *# Additional offset for pawn*

**if** piece == 'pawn':

screen.blit(white\_pawn **if** colour == 'red' **else** black\_pawn, (locations[j][0] \* 100 + pawn\_offset\_x, locations[j][1] \* 100 + pawn\_offset\_y)) *# Drawing the pawns. The red legal move highlights are associated with white which is why I have used the string 'red' to mean white.*

**else**:

screen.blit(white\_images[index] **if** colour == 'red' **else** black\_images[index], (locations[j][0] \* 100 + offset\_x, locations[j][1] \* 100 + offset\_y)) *# Drawing the pieces.*

*# Draw selection rectangle*

**if** (turn\_step < 2 **and** i == 0) **or** (turn\_step >= 2 **and** i == 1):

**if** selection == j:

pygame.draw.rect(screen, colour, [locations[j][0] \* 100 + 1, locations[j][1] \* 100 + 1, 100, 100], 2)

**def** check\_options(pieces, locations, turn, castling\_moves, check):

*#This function is responsible for fetching all of the legal moves in a given position.*

*# Initially there were global variables incorporated in the design, which came from the tutorial but I removed them and edited the functionality accordingly so as to incorporate better coding design.*

all\_moves\_list = []

new\_castling\_moves = castling\_moves.copy()

**for** i **in** range(len(pieces)):

location = locations[i]

piece = pieces[i]

**if** piece == 'pawn':

moves\_list = check\_pawn(location, turn) *# get the legal pawn moves.*

**elif** piece == 'rook':

moves\_list = check\_rook(location, turn) *# get the legal rook moves.*

**elif** piece == 'knight':

moves\_list = check\_knight(location, turn) *# get the legal knight moves.*

**elif** piece == 'bishop':

moves\_list = check\_bishop(location, turn) *# get the legal bishop moves.*

**elif** piece == 'queen':

moves\_list = check\_queen(location, turn) *# get the legal queen moves.*

**elif** piece == 'king':

moves\_list, new\_castling\_moves = check\_king(location, turn, check) *# get the legal king moves. store the castling moves as a separate list*

all\_moves\_list.append(moves\_list) *# combine all of the moves*

**return** all\_moves\_list, new\_castling\_moves

**def** get\_colour\_lists(colour): *# To determine in a certain turn, which colour is your own pieces and which side is your opponent's pieces.*

*# For example, if you have the white pieces, your own pieces will be white and the enemy pieces will be black.*

**if** colour == 'white':

**return** black\_locations, white\_locations

**else**:

**return** white\_locations, black\_locations

**def** check\_king(position, colour, check):

*#This function is responsible for fetching all of the legal moves that a king piece can make.*

*#The tutorial I followed had global variables here but in order to demonstrate better coding style, I removed them from the code.*

moves\_list = [] *#Stores all of the legal king moves.*

castle\_moves, check = check\_castling(check) *#Gets all of the castling moves that a king piece can make*

enemies\_list, friends\_list = get\_colour\_lists(colour)

**for** x\_pos **in** [-1, 0, 1]: *# A king can move one square in any direction. This for loop focusses on the x position.*

**for** y\_pos **in** [-1, 0, 1]: *# This nested for loops looks at the y position*

**if** x\_pos == y\_pos == 0: *#If they are both equal to 0, the king makes no moves.*

**pass**

target = (position[0] + x\_pos, position[1] + y\_pos)

**if** 0 <= target[0] <= 7 **and** 0 <= target[1] <= 7 **and** target **not** **in** friends\_list: *#Here, we make sure that the final end position the king ends up on is on the board (i.e. on a square with coordinates between 1 and 7 for both the x and y coordinates).*

moves\_list.append(target)

**return** moves\_list, castle\_moves

**def** check\_queen(position, colour):

*# This function is responsble for fetching the legal moves that can be made by a queen in a given position.*

*# A queen simply combines the functionalities of a rook and a bishop so we can simply reuse those functions.*

moves\_list = check\_bishop(position, colour)

second\_list = check\_rook(position, colour)

**for** i **in** range(len(second\_list)):

moves\_list.append(second\_list[i]) *# Additing the legal rook moves into one list.*

**return** moves\_list

**def** check\_bishop(position, colour):

*#This function is responsible for fetching the legal moves that can be made by a bishop in a given position.*

*#The bishop is able to move along the diagonals through as many squares as desired until there is a blockade in the diagonal that is being pursued.*

moves\_list = []

directions = [(1, -1), (-1, -1), (1, 1), (-1, 1)] *#These are the basic direction vectors of the movement without any chain.*

enemies\_list, friends\_list = get\_colour\_lists(colour)

**for** x\_pos, y\_pos **in** directions:

current\_x, current\_y = position *#Keeps track of starting position.*

chain = 1

keep\_checking = True

**while** keep\_checking **and** 0 <= current\_x + chain \* x\_pos <= 7 **and** 0 <= current\_y + chain \* y\_pos <= 7: *#Ensures that the chain stays on the board.*

new\_position = (current\_x + chain \* x\_pos, current\_y + chain \* y\_pos) *#The new position from moving in a certain direction.*

**if** new\_position **not** **in** friends\_list: *#Ensuring that our own pieces aren't already on that speciifc square.*

moves\_list.append(new\_position)

**if** new\_position **in** enemies\_list:

keep\_checking = False *# We have reached a 'blockade' position so we simply append the move.*

chain += 1

**else**:

keep\_checking = False

**return** moves\_list

**def** check\_rook(position, colour):

*#This function is responsible for fetching all of the legal rook moves in a given position.*

*#The rook works in a very similar way to the bishop except that the rook moves horizontally or vertically.*

moves\_list = []

enemies\_list, friends\_list = get\_colour\_lists(colour)

directions = [(0, 1), (0, -1), (1, 0), (-1, 0)] *# down, up, right, left*

**for** x\_pos, y\_pos **in** directions:

start\_x, start\_y = position

chain = 1

continue\_checking = True

**while** continue\_checking **and** 0 <= start\_x + chain \* x\_pos <= 7 **and** 0 <= start\_y + chain \* y\_pos <= 7: *# The loop will continue to add the basc vectors in the direction list until one of the positions isn't present on the board.*

new\_position = (start\_x + chain \* x\_pos, start\_y + chain \* y\_pos)

**if** new\_position **not** **in** friends\_list:

moves\_list.append(new\_position)

**if** new\_position **in** enemies\_list:

continue\_checking = False *# We can stop checking for new positions as the file has been blockaded.*

chain += 1

**else**:

continue\_checking = False *# our own pieces are blocking our path.*

**return** moves\_list

**def** is\_valid\_square(x, y):

*# returns a Boolean value to indicate whether the square is valid*

**return** 0 <= x <= 7 **and** 0 <= y <= 7

**def** check\_pawn(position, colour):

moves\_list = []

**if** colour == 'white':

**if** (position[0], position[1] + 1) **not** **in** white\_locations **and** \

(position[0], position[1] + 1) **not** **in** black\_locations **and** position[1] < 7:

moves\_list.append((position[0], position[1] + 1))

*# indent the check for two spaces ahead, so it is only checked if one space ahead is also open*

**if** (position[0], position[1] + 2) **not** **in** white\_locations **and** \

(position[0], position[1] + 2) **not** **in** black\_locations **and** position[1] == 1:

moves\_list.append((position[0], position[1] + 2))

**if** (position[0] + 1, position[1] + 1) **in** black\_locations:

moves\_list.append((position[0] + 1, position[1] + 1))

**if** (position[0] - 1, position[1] + 1) **in** black\_locations:

moves\_list.append((position[0] - 1, position[1] + 1))

*# add en passant move checker*

**if** (position[0] + 1, position[1] + 1) == black\_ep:

moves\_list.append((position[0] + 1, position[1] + 1))

**if** (position[0] - 1, position[1] + 1) == black\_ep:

moves\_list.append((position[0] - 1, position[1] + 1))

**else**:

**if** (position[0], position[1] - 1) **not** **in** white\_locations **and** \

(position[0], position[1] - 1) **not** **in** black\_locations **and** position[1] > 0:

moves\_list.append((position[0], position[1] - 1))

*# indent the check for two spaces ahead, so it is only checked if one space ahead is also open*

**if** (position[0], position[1] - 2) **not** **in** white\_locations **and** \

(position[0], position[1] - 2) **not** **in** black\_locations **and** position[1] == 6:

moves\_list.append((position[0], position[1] - 2))

**if** (position[0] + 1, position[1] - 1) **in** white\_locations:

moves\_list.append((position[0] + 1, position[1] - 1))

**if** (position[0] - 1, position[1] - 1) **in** white\_locations:

moves\_list.append((position[0] - 1, position[1] - 1))

*# add en passant move checker*

**if** (position[0] + 1, position[1] - 1) == white\_ep:

moves\_list.append((position[0] + 1, position[1] - 1))

**if** (position[0] - 1, position[1] - 1) == white\_ep:

moves\_list.append((position[0] - 1, position[1] - 1))

**return** moves\_list

**def** check\_knight(position, colour):

*# This function is responsible for fetching every legal move that can be made by a knight in a given position.*

moves\_list = []

enemies\_list, friends\_list = get\_colour\_lists(colour)

*# 8 squares to check for knights, they can go two squares in one direction and one in another*

targets = [(1, 2), (1, -2), (2, 1), (2, -1), (-1, 2), (-1, -2), (-2, 1), (-2, -1)]

**for** move **in** targets:

target = (position[0] + move[0], position[1] + move[1])

**if** target **not** **in** friends\_list **and** 0 <= target[0] <= 7 **and** 0 <= target[1] <= 7:

moves\_list.append(target)

**return** moves\_list

**def** get\_options\_list(white\_options, black\_options, turn\_step):

*# Works out whose turn it is so that we don't have to work out the legal moves of both the black and the white pieces for a given turn.*

**return** white\_options **if** turn\_step < 2 **else** black\_options

**def** check\_valid\_moves(white\_options, black\_options, turn\_step, selection):

*# check for valid moves for just selected piece*

options\_list = get\_options\_list(white\_options, black\_options, turn\_step)

valid\_options = options\_list[selection]

**return** valid\_options

**def** determineColour(turn\_step):

*# This is a simple function that works out the colour of the squares highlighter based off whose turn it is, which is signified by the turn\_step variable.*

colour = 'red' **if** turn\_step < 2 **else** 'blue'

**return** colour

**def** draw\_valid(moves, turn\_step):

*# After checking legality, we draw the moves on the board using this function.*

colour = determineColour(turn\_step)

**for** i **in** range(len(moves)):

pygame.draw.circle(screen, colour, (moves[i][0] \* 100 + 50, moves[i][1] \* 100 + 50), 5)

**def** determineCheckColour(turn\_step):

*# We work out which king is in check. If it is the white king, it will have a red piece highlight. If it is the black king, it will have a blue piece highlight.*

*# The highlights will flash to indicate that you are in check because I had problems with implementing an algorithm to draw the moves that either got out of the check, blocked the check or had the potential to capture the offending piece.*

**if** turn\_step < 2: *# White's king is in check*

colour = 'dark red'

**else**:

colour = 'dark blue' *# Black's king is in check*

**return** colour

**def** draw\_check(turn\_step, counter):

*# This function will draw a flashing highlight around the king that is in check when appropriate.*

king\_index, king\_location = None, None

**if** turn\_step < 2: *# If it's white's turn*

**if** 'king' **in** white\_pieces:

king\_index = white\_pieces.index('king')

king\_location = white\_locations[king\_index]

opponent\_options = black\_options

**else**:

**return** False *# the king isn't in check so we just return back to the game.*

**else**:

**if** 'king' **in** black\_pieces:

king\_index = black\_pieces.index('king')

king\_location = black\_locations[king\_index]

opponent\_options = white\_options

**else**:

**return** False

**for** i **in** range(len(opponent\_options)):

**if** king\_location **in** opponent\_options[i]:

**if** counter < 15:

colour = determineCheckColour(turn\_step)

pygame.draw.rect(screen, colour, [king\_location[0] \* 100 + 1, king\_location[1] \* 100 + 1, 100, 100], 5)

**return** True

**return** False *# We return False as a default if none of the other conditions are met.*

**def** draw\_game\_over(winner):

*# When the game is over, this function will output the message saying who won and will also give the option to restart the game.*

pygame.draw.rect(screen, 'black', [200, 200, 400, 70])

**if** winner == "white" **or** winner == "black":

screen.blit(font.render(f'{winner} won the game!', True, 'white'), (210, 210))

**else**:

screen.blit(font.render("Draw by stalemate!", True, 'white'), (210, 210))

screen.blit(font.render(f'Press ENTER to Restart!', True, 'white'), (210, 240))

**def** check\_en\_passant(turn\_step, old\_coords, new\_coords):

*# This function is responsible for checking whether en passant is legal or not.*

**if** turn\_step <= 1: *# White's en passant*

index = white\_locations.index(old\_coords) *# Get the old pawn position.*

ep\_coords = (new\_coords[0], new\_coords[1] - 1) *# Convert to the new pawn position after the move has been made and selected.*

piece = white\_pieces[index]

**else**:

index = black\_locations.index(old\_coords)

ep\_coords = (new\_coords[0], new\_coords[1] + 1)

piece = black\_pieces[index]

**if** piece == 'pawn' **and** abs(old\_coords[1] - new\_coords[1]) > 1:

*# if piece was pawn and moved two spaces, return EP coords as defined above*

**pass**

**else**:

ep\_coords = (100, 100) *# Default values if none of the checks return as true (for example, when we have a piece that has been selected. )*

**return** ep\_coords

**def** check\_castling(check):

*# # king must not currently be in check, neither the rook nor king has moved previously, nothing between*

*# and the king does not pass through or finish on an attacked piece*

*# Store each valid castle move as [((king\_coords), (rook\_coords))]*

castle\_moves = []

rook\_indexes = []

rook\_locations = []

king\_index = 0

king\_pos = (0, 0)

*# Determine if it's White's or Black's turn based on the turn step*

**if** turn\_step > 1: *# White's turn*

*# Iterate over White pieces to find rooks and the king*

**for** i **in** range(len(white\_pieces)):

**if** white\_pieces[i] == 'rook': *# Finding whether a rook exists.*

rook\_indexes.append(white\_moved[i])

rook\_locations.append(white\_locations[i])

**if** white\_pieces[i] == 'king':

king\_index = i

king\_pos = white\_locations[i]

*# Check conditions for castling on White's side*

**if** **not** white\_moved[king\_index] **and** False **in** rook\_indexes **and** **not** check:

**for** i **in** range(len(rook\_indexes)):

castle = True

**if** rook\_locations[i][0] > king\_pos[0]:

empty\_squares = [(king\_pos[0] + 1, king\_pos[1]), (king\_pos[0] + 2, king\_pos[1]), (king\_pos[0] + 3, king\_pos[1])]

**else**:

empty\_squares = [(king\_pos[0] - 1, king\_pos[1]), (king\_pos[0] - 2, king\_pos[1])]

**for** j **in** range(len(empty\_squares)):

*# Check if there are any pieces on the empty squares, or if the rook is moved, or if it's under attack*

**if** empty\_squares[j] **in** white\_locations **or** empty\_squares[j] **in** black\_locations **or** \

empty\_squares[j] **in** black\_options **or** rook\_indexes[i]:

castle = False

*# If all conditions are met, add the castle move to the list*

**if** castle:

castle\_moves.append((empty\_squares[1], empty\_squares[0]))

**else**:

*# Iterate over Black pieces to find rooks and the king*

**for** i **in** range(len(black\_pieces)):

**if** black\_pieces[i] == 'rook':

rook\_indexes.append(black\_moved[i])

rook\_locations.append(black\_locations[i])

**if** black\_pieces[i] == 'king':

king\_index = i

king\_pos = black\_locations[i]

*# Check conditions for castling on Black's side*

**if** **not** black\_moved[king\_index] **and** False **in** rook\_indexes **and** **not** check:

**for** i **in** range(len(rook\_indexes)):

castle = True

**if** rook\_locations[i][0] > king\_pos[0]:

empty\_squares = [(king\_pos[0] + 1, king\_pos[1]), (king\_pos[0] + 2, king\_pos[1]),(king\_pos[0] + 3, king\_pos[1])]

**else**:

empty\_squares = [(king\_pos[0] - 1, king\_pos[1]), (king\_pos[0] - 2, king\_pos[1])]

**for** j **in** range(len(empty\_squares)):

*# Check if there are any pieces on the empty squares, or if the rook is moved, or if it's under attack*

**if** empty\_squares[j] **in** white\_locations **or** empty\_squares[j] **in** black\_locations **or** \

empty\_squares[j] **in** white\_options **or** rook\_indexes[i]:

castle = False

*# If all conditions are met, add the castle move to the list*

**if** castle:

castle\_moves.append((empty\_squares[1], empty\_squares[0]))

**return** castle\_moves, check

**def** draw\_castling(turn\_step, moves):

*# This function is responsible for drawing the castling move.*

colour = determineColour(turn\_step)

**for** move **in** moves:

king\_x, king\_y = move[0][0] \* 100 + 50, move[0][1] \* 100 + 70

rook\_x, rook\_y = move[1][0] \* 100 + 50, move[1][1] \* 100 + 70

king\_center = (king\_x, king\_y)

rook\_center = (rook\_x, rook\_y)

king\_render\_pos = (king\_x - 20, king\_y - 10)

rook\_render\_pos = (rook\_x - 20, rook\_y - 10)

pygame.draw.circle(screen, colour, king\_center, 8)

screen.blit(font.render('king', True, 'black'), king\_render\_pos)

pygame.draw.circle(screen, colour, rook\_center, 8)

screen.blit(font.render('rook', True, 'black'), rook\_render\_pos)

pygame.draw.line(screen, colour, king\_center, rook\_center, 2)

*# add pawn promotion*

**def** check\_promotion():

*# This function is responsible for checking whether a white pawn has reached the eighth rank or a black pawn has reached the first rank.*

pawn\_indexes = []

white\_promotion = False

black\_promotion = False

promote\_index = 100

**for** i **in** range(len(white\_pieces)):

**if** white\_pieces[i] == 'pawn':

pawn\_indexes.append(i)

**for** i **in** range(len(pawn\_indexes)):

**if** white\_locations[pawn\_indexes[i]][1] == 7:

white\_promotion = True

promote\_index = pawn\_indexes[i]

pawn\_indexes = []

**for** i **in** range(len(black\_pieces)):

**if** black\_pieces[i] == 'pawn':

pawn\_indexes.append(i)

**for** i **in** range(len(pawn\_indexes)):

**if** black\_locations[pawn\_indexes[i]][1] == 0:

black\_promotion = True

promote\_index = pawn\_indexes[i]

**return** white\_promotion, black\_promotion, promote\_index

**def** draw\_promotion():

*# When a pawn reaches the eighth or first rank, a menu will be drawn upon completion of the move, asking the user to select a piece to promote the pawn to.*

pygame.draw.rect(screen, 'dark gray', [800, 0, 200, 420])

colour = 'white' **if** white\_promote **else** 'black'

promotions = white\_promotions **if** white\_promote **else** black\_promotions

**for** i, piece **in** enumerate(promotions):

index = piece\_list.index(piece)

piece\_image = white\_images[index] **if** colour == 'white' **else** black\_images[index]

screen.blit(piece\_image, (860, 5 + 100 \* i))

pygame.draw.rect(screen, colour, [800, 0, 200, 420], 8)

**def** check\_promo\_select():

*# Work out what the user wants to promote to.*

mouse\_pos = pygame.mouse.get\_pos()

left\_click = pygame.mouse.get\_pressed()[0]

x\_pos = mouse\_pos[0] // 100

y\_pos = mouse\_pos[1] // 100

*# Check if the promotion index is not None and the mouse position is within the promotion area*

**if** promo\_index **is** **not** None **and** y\_pos < 4 **and** left\_click **and** x\_pos >= 8:

*# Check if it's a white promotion and the indices are within bounds*

**if** white\_promote **and** 0 <= promo\_index < len(white\_pieces) **and** 0 <= y\_pos < len(white\_promotions):

white\_pieces[promo\_index] = white\_promotions[y\_pos]

*# Check if it's a black promotion and the indices are within bounds*

**elif** black\_promote **and** 0 <= promo\_index < len(black\_pieces) **and** 0 <= y\_pos < len(black\_promotions):

black\_pieces[promo\_index] = black\_promotions[y\_pos]

**def** getBgCol():

*# Get the background colour that the user wants.*

*# Like on lichess.org and chess.com, the background colour is customisable*

colours = ["red", "orange", "yellow", "green", "light green", "dark green", "blue", "light blue", "dark blue", "pink", "purple", "brown", "grey", "white"]

valid = False

**print**("**\n**Please select a background colour.")

**while** valid == False:

bgcol = input(">").lower()

**try**:

**if** bgcol **not** **in** colours: *# if a valid colour isn't entered*

**raise** ColourError("That colour is not available. These are the colours that are available:")

valid = True

**except** ColourError:

**print**("Please enter a valid colour. You can't select black because you will not be able to see the captured pieces.")

**return** bgcol

**def** selectPieceSet():

*# This function is responsible for getting the user to select a piece set. It is running recursively.*

*# Dictionary below deals with selecting a piece set. The key value is the number and the pair value is the folder name in which the assets are being stored.*

piece\_sets = {

1: "basic",

2: "merida",

3: "3D",

4: "anarcandy",

}

**print**("Select a piece set for your game.")

**print**("These are your options:")

**for** number, piece\_set **in** piece\_sets.items():

**print**(f"{number}: {piece\_set}")

time.sleep(0.25) *#Improved readability.*

**while** True:

**try**:

selected\_number = int(input(">"))

**if** selected\_number **in** piece\_sets:

selected\_piece\_set = piece\_sets[selected\_number]

**print**("OK, "+str(selected\_piece\_set)+" has been selected.")

**return** selected\_piece\_set

**else**:

**print**("Please enter a number between 1 and 4.")

**except** ValueError: *#To deal with any rogue input values.*

**print**("Please enter a valid integer between 1 and 4.")

*################################################################ - UI Functions - #################################################################*

**class** CreateAccountScreen(tk.Frame): *#Inheritance and Polymorphism from the base class.*

*# This is the first screen that will be displayed to the user upon execution of the code.*

**def** \_\_init\_\_(self, master=None):

super().\_\_init\_\_(master, bg="black") *#Overriding the default.*

self.master = master

self.grid(sticky="nsew")

self.\_\_create\_widgets()

**def** \_\_create\_widgets(self):

*# Create the "Create account" label with a bigger font*

create\_account\_label = tk.Label(self, text="Create account", font=("Arial", 36), fg="orange", bg="black")

create\_account\_label.grid(row=1, pady=(5, 15), padx=(125, 50), columnspan=4)

*# Create the "Username:" label*

username\_label = tk.Label(self, text="Username:", font=("Arial", 14), fg="white", bg="black")

username\_label.grid(row=2, column=0, sticky='e', padx=(3, 0), pady=(0, 3))

*# Create the input box to the right of the "Username" label with red text*

self.\_\_username\_input = tk.Entry(self, font=("Arial", 14), bg="dark blue", fg="red")

self.\_\_username\_input.grid(row=2, column=1, padx=(0, 3), pady=(0, 3))

*# Bind the focus and focus-out events to change the background colour*

self.\_\_username\_input.bind("<FocusIn>", self.\_\_on\_entry\_focus\_username)

self.\_\_username\_input.bind("<FocusOut>", self.\_\_on\_entry\_focus\_out\_username)

self.\_\_password\_label = tk.Label(self, text="Password:", font=("Arial", 14), fg="white", bg="black")

self.\_\_password\_label.grid(row=3, column=0, sticky='e', padx=(3, 0), pady=(0, 3))

*# Password Entry*

self.\_\_password\_input = tk.Entry(self, font=("Arial", 14), bg="dark blue", fg="red", show="\*")

self.\_\_password\_input.grid(row=3, column=1, padx=(0, 3), pady=(0, 3))

self.\_\_password\_input.bind("<FocusIn>", self.\_\_on\_entry\_focus\_password)

self.\_\_password\_input.bind("<FocusOut>", self.\_\_on\_entry\_focus\_out\_password)

*# Create the "Submit" button with a red background to switch to the Main Menu screen*

submit\_button = tk.Button(self, text="Create", command=self.\_\_submit\_details, font=("Arial", 16), bg="red", fg="blue")

submit\_button.grid(row=4, columnspan=4, pady=10)

*# Add an error label to display the error message*

self.\_error\_label = tk.Label(self, text="", fg="red", bg="black")

self.\_error\_label.grid(row=6, columnspan=4)

*# Add an image 200 pixels below the "Submit" button*

logo\_image = Image.open("logo.png")

logo\_image = logo\_image.resize((400, 200), Image.LANCZOS)

logo\_photo = ImageTk.PhotoImage(logo\_image)

self.\_logo\_label = tk.Label(self, image=logo\_photo, bg="black")

self.\_logo\_label.image = logo\_photo

self.\_logo\_label.grid(row=7, columnspan=4, pady=10)

*# Create a "Log In" button to switch to the LogInScreen*

login\_button = tk.Button(self, text="Log In", command=self.\_\_switch\_to\_login\_screen, font=("Arial", 16), bg="red", fg="blue")

login\_button.grid(row=8, columnspan=4, pady=10)

quit\_button = tk.Button(self, text="Quit", command=self.\_\_quit\_program, font=("Arial", 16), bg="red", fg="blue")

quit\_button.grid(row=7, column=7, pady=15)

**def** \_\_on\_entry\_focus\_password(self, event): *#The password field will change colour to light blue when it is clicked.*

event.widget.config(bg="#7ab8ff")

**def** \_\_on\_entry\_focus\_out\_password(self, event): *# The password field will be dark blue when it is yet to be clicked.*

event.widget.config(bg="dark blue")

**def** \_\_on\_entry\_focus\_username(self, event): *#The password field will change colour to light blue when it is clicked.*

event.widget.config(bg="#7ab8ff")

**def** \_\_on\_entry\_focus\_out\_username(self, event): *# The password field will be dark blue when it is yet to be clicked.*

event.widget.config(bg="dark blue")

**def** \_\_submit\_details(self):

*# This function is responsible for adding the details to the database. If the username already exists in the database, an error will output informing the user to create a new one.*

connection = sqlite3.connect("LoginDetails.db")

cursor = connection.cursor()

*# A table called PasswordDetails with the passwordID and the hah value of the password being stored.*

cursor.execute('''

CREATE TABLE IF NOT EXISTS PasswordDetails (

passwordID INTEGER PRIMARY KEY AUTOINCREMENT,

hashed\_password TEXT

)

''')

*# A table called IDDetails which stored the userID and passwordID.*

cursor.execute('''

CREATE TABLE IF NOT EXISTS IDDetails (

userID INTEGER PRIMARY KEY AUTOINCREMENT,

passwordID INTEGER,

FOREIGN KEY(passwordID) REFERENCES PasswordDetails(passwordID)

)

''')

*# A tabel called Users, which stores the UserID and the username.*

cursor.execute('''

CREATE TABLE IF NOT EXISTS UsernameDetails (

userID INTEGER PRIMARY KEY,

username TEXT UNIQUE,

FOREIGN KEY(userID) REFERENCES IDDetails(userID)

)

''')

**if** len(self.\_\_username\_input.get()) < 6: *#Checks if the username is less than 6 characters*

messagebox.showinfo("Error","Your username needs to be at least 6 characters!")

**elif** len(self.\_\_username\_input.get()) > 15: *# Checks whether the username is greater than 15 characters.*

messagebox.showinfo("Error", "Your username needs to be less than or equal to 15 characters!")

**elif** len(self.\_\_password\_input.get()) < 6: *# Added security measure of checking password length.*

messagebox.showinfo("Error", "Your password needs to be at least 6 characters!")

**else**:

*# Hash the password*

hashed\_password = self.\_\_HashPassword(self.\_\_password\_input.get())  *# Password gets hashed.*

*# Check if username already exists*

cursor.execute('SELECT \* FROM UsernameDetails WHERE username = ?', (self.\_\_username\_input.get(),))

**if** cursor.fetchone(): *# Does the username exist*

messagebox.showerror("Error", "Username already exists. Directing you to the login screen...") *#yes*

self.\_\_switch\_to\_login\_screen()

**else**:

*# Insert data into PasswordDetails table*

cursor.execute('INSERT INTO PasswordDetails (hashed\_password) VALUES (?)', (hashed\_password,)) *#no*

passwordID = cursor.lastrowid

*# Insert data into IDDetails table*

cursor.execute('INSERT INTO IDDetails (userID, passwordID) VALUES (?, ?)', (cursor.lastrowid, passwordID))

*# Get the userID of the inserted row*

userID = cursor.lastrowid

*# Insert data into Users table*

cursor.execute('INSERT INTO UsernameDetails (userID, username) VALUES (?, ?)', (userID, self.\_\_username\_input.get()))

*# Commit changes and close connection*

messagebox.showinfo("Success", "Account created successfully!")

connection.commit()

connection.close()

*# Direct to Main Menu Screen*

self.\_\_switch\_to\_main\_menu\_screen()

**def** \_\_HashPassword(self, password):

*# This function is responsible for hashing the password to improve the security of the system and for additional verification and authentication processes before allowing the user to enter the system.*

rt = 0

**for** i **in** range(0, len(password)-1):

rt += ord(password[i])

**return** rt

**def** \_\_switch\_to\_main\_menu\_screen(self):

*# This function is responsible for switching to the Main Menu Screen*

self.master.switch\_frame(MainMenuScreen)

**def** \_\_switch\_to\_login\_screen(self):

*# Thie function is responsible for switching to the Login Screen*

self.master.switch\_frame(LogInScreen)

**def** \_\_quit\_program(self):

*# This function will quit the program if the user clicks on the Quit button*

self.master.destroy()

**class** LogInScreen(tk.Frame): *#Inheritance and Polymorphism from the tk.Frame class.*

*# If an account has already been created with the system, clicking on the LogIn button will direct them to this screen where they will be able to login and see whether their account exists or not.*

**def** \_\_init\_\_(self, master=None):

super().\_\_init\_\_(master, bg="black")

self.master = master

self.grid(sticky="nsew")

self.\_\_create\_widgets()

**def** \_\_create\_widgets(self):

*# Create the "Log In" label with a bigger font*

login\_label = tk.Label(self, text="Log In", font=("Arial", 36), fg="orange", bg="black")

login\_label.grid(row=1, pady=(100, 15), columnspan=4)

*# Create the "Username:" label*

username\_label = tk.Label(self, text="Username:", font=("Arial", 14), fg="white", bg="black")

username\_label.grid(row=2, column=0, sticky='e', padx=(3, 0), pady=(0, 3))

*# Create the input box for the username*

self.\_\_username\_input = tk.Entry(self, font=("Arial", 14), bg="dark blue", fg="red")

self.\_\_username\_input.grid(row=2, column=1, padx=(0, 3), pady=(0, 3))

self.\_\_username\_input.bind("<FocusIn>", self.\_\_on\_entry\_focus\_username)

self.\_\_username\_input.bind("<FocusOut>", self.\_\_on\_entry\_focus\_out\_username)

*# Create the "Login" button with a red background to check the username*

login\_button = tk.Button(self, text="Log In", command=self.\_\_check\_login, font=("Arial", 16), fg="blue", bg="red")

login\_button.grid(row=5, columnspan=4, pady=10)

self.\_\_password\_label = tk.Label(self, text="Password:", font=("Arial", 14), fg="white", bg="black")

self.\_\_password\_label.grid(row=3, column=0, sticky='e', padx=(3, 0), pady=(0, 3))

self.\_\_password\_input = tk.Entry(self, font=("Arial", 14), bg="dark blue", fg="red", show="\*")

self.\_\_password\_input.grid(row=3, column=1, padx=(0, 3), pady=(0, 3))

self.\_\_password\_input.bind("<FocusIn>", self.\_\_on\_entry\_focus\_password)

self.\_\_password\_input.bind("<FocusOut>", self.\_\_on\_entry\_focus\_out\_password)

*# Create an error label to display the error message*

self.\_\_error\_label = tk.Label(self, text="", fg="red", bg="black")

self.\_\_error\_label.grid(row=4, columnspan=4)

*# Add an image below the error label*

logo\_image = Image.open("logo.png")

logo\_image = logo\_image.resize((300, 150), Image.LANCZOS)

logo\_photo = ImageTk.PhotoImage(logo\_image)

self.\_\_logo\_label = tk.Label(self, image=logo\_photo, bg="black")

self.\_\_logo\_label.image = logo\_photo

self.\_\_logo\_label.grid(row=6, columnspan=4, pady=10)

back\_button = tk.Button(self, text="Back", command=self.master.go\_back, font=("Arial", 16), fg="blue", bg="red")

back\_button.grid(row=2, column=10, pady=10)

home\_button = tk.Button(self, text="Home", command=self.\_\_switch\_to\_create\_account, font=("Arial", 16), fg="blue", bg="red")

home\_button.grid(row=3, column=10, pady=10)

**def** \_\_HashPassword(self, password):

*#This is the same password hashing function as in the CreateAccountScreen class.*

rt = 0

**for** i **in** range(0, len(password)-1):

rt += ord(password[i])

**return** rt

**def** \_\_switch\_to\_create\_account(self):

*# This function switches the screen back to the CreateAccountScreen class*

self.master.switch\_frame(CreateAccountScreen)

**def** \_\_check\_login(self):

**try**:

*# Connect to the database*

connection = sqlite3.connect("LoginDetails.db")

cursor = connection.cursor()

*# Check if UsernameDetails table exists*

cursor.execute("SELECT name FROM sqlite\_master WHERE type='table' AND name='UsernameDetails'")

**if** cursor.fetchone():

*# Get the entered password*

entered\_password = self.\_\_password\_input.get()

*# Check 1: Does the username exist in the database*

cursor.execute('SELECT \* FROM UsernameDetails WHERE username = ?', (self.\_\_username\_input.get(),))

user\_data = cursor.fetchone()

**if** user\_data:

*# Check 2: Compare hashed passwords in IDDetails table*

hashed\_entered\_password = self.\_\_HashPassword(entered\_password)

cursor.execute('''

SELECT hashed\_password

FROM PasswordDetails

WHERE passwordID = ?

''', (user\_data[1],)) *# Assuming passwordID is in the second column of IDDetails*

hashed\_db\_password = cursor.fetchone()

*# Check 3: Check for plaintext password (existing check)*

**if** entered\_password == user\_data[1]: *# Assuming plaintext password is stored in the second column*

*# Perform additional checks if needed*

*# Successful login*

messagebox.showinfo("Success", "Logging In...")

self.\_\_switch\_to\_main\_menu()

**elif** hashed\_db\_password **and** hashed\_entered\_password == hashed\_db\_password[0]:

*# Successful login*

messagebox.showinfo("Success", "Logging In...")

self.\_\_switch\_to\_main\_menu()

**else**:

*# Incorrect password*

messagebox.showerror("Error", "Invalid password!")

**else**:

*# Invalid username*

messagebox.showerror("Error", "Invalid username!")

**else**:

*# UsernameDetails table doesn't exist*

messagebox.showerror("Error", "No usernames have been created!")

**except** sqlite3.Error **as** e:

*# Handle SQLite errors*

messagebox.showerror("Error", f"SQLite error: {e}")

**finally**:

*# Close the database connection*

**if** connection:

connection.close()

**def** \_\_on\_entry\_focus\_password(self, event):

event.widget.config(bg="#7ab8ff")

**def** \_\_on\_entry\_focus\_out\_password(self, event):

event.widget.config(bg="dark blue")

**def** \_\_on\_entry\_focus\_username(self, event):

event.widget.config(bg="#7ab8ff")

**def** \_\_on\_entry\_focus\_out\_username(self, event):

event.widget.config(bg="dark blue")

**def** \_\_switch\_to\_main\_menu(self):

*# This function is responsible for switching to the Main Menu.*

self.master.switch\_frame(MainMenuScreen)

**class** MainMenuScreen(tk.Frame):

*# This class displays the main menu and the options that are available for the user.*

**def** \_\_init\_\_(self, master=None):

super().\_\_init\_\_(master, bg="black")

self.master = master

self.grid(sticky="nsew")

self.\_\_create\_widgets()

**def** \_\_create\_widgets(self):

main\_menu\_label = tk.Label(self, text="Main Menu", font=("Arial", 24), fg="orange", bg="black")

main\_menu\_label.grid(row=0, column = 10, pady=(100, 20), columnspan=10)

*# Create a "Standard Chess" button*

standard\_chess\_button = tk.Button(self, text="Standard Chess", command=self.\_\_return\_standard\_chess, font=("Arial", 16), fg="blue", bg="red", height = 2) *# Normal game of chess player vs player*

standard\_chess\_button.grid(row=1, column=12, padx=10, pady=15)

big\_bang\_variant\_button = tk.Button(self, text="Big Bang Variant", command=self.\_\_return\_big\_bang\_variant, font=("Arial", 16), fg="blue", bg="red", height = 2) *# My own Big Bang variant*

big\_bang\_variant\_button.grid(row=2, column=12, padx = 10, pady=15)

lichess\_stats\_\_button = tk.Button(self, text="View Lichess Stats", command=self.\_\_return\_get\_lichess\_data, font=("Arial", 16), fg="blue", bg="red", height = 2) *#Viewing lichess stats but this code is on a different file called ViewLichessStats.py*

lichess\_stats\_\_button.grid(row=4, column=12, padx=10, pady=10)

self.\_\_message\_label = tk.Label(self, text="", font=("Arial", 16), fg="red", bg="black") *#Potential for dealing with any errors, should they arise. This could also be used to display any messages like the need to go onto a different file.*

self.\_\_message\_label.grid(row= 10, column=10, pady=10)

back\_button = tk.Button(self, text="Back", command=self.master.go\_back, font=("Arial", 16), fg="blue", bg="red") *# This function will pop the last screen from the stack and then return the user to that screen.*

back\_button.grid(row=5, column=8, pady=10)

**def** \_\_return\_standard\_chess(self):

*# Return a value of 1 to show that the user wants to play the standard game of chess.*

self.\_\_quit\_program(1)

**def** \_\_return\_big\_bang\_variant(self):

*# Return a value of 2 to show that the user wants to play the variant.*

self.\_\_quit\_program(2)

**def** \_\_return\_get\_lichess\_data(self):

*# Output the error message if the View Lichess Stats button is clicked*

self.\_\_message\_label.config(text="To view the data, run the code in ViewLichessStats.py")

**def** \_\_quit\_program(self, value):

self.master.return\_value = value

self.master.destroy()

**def** \_\_switch\_to\_create\_account(self):

self.master.switch\_frame(CreateAccountScreen)

**def** \_\_switch\_to\_main\_menu(self):

self.master.switch\_frame(MainMenuScreen)

**class** App(tk.Tk):

*# Basic interface is initialised before any changes are made to it.*

**def** \_\_init\_\_(self, start\_frame = CreateAccountScreen):

super().\_\_init\_\_()

self.geometry("800x700")

self.title("Create Account")

self.configure(bg="black")

self.grid\_rowconfigure(0, weight=1)

self.grid\_columnconfigure(0, weight=1)

self.return\_value = None

self.current\_frame = None

self.\_stack = Stack(10) *# Initialize the stack. Association* *aggregation shown here.*

self.is\_first\_switch = True

self.switch\_frame(start\_frame)

**def** switch\_frame(self, frame\_class, going\_back=False):

*# This method defines how the screens switch between each other.*

**if** **not** going\_back **and** self.current\_frame:

self.\_stack.push(self.current\_frame.\_\_class\_\_)

new\_frame = frame\_class(self)

**if** self.current\_frame **is** **not** None:

self.current\_frame.destroy()

self.current\_frame = new\_frame

self.current\_frame.grid(row=0, column=0, sticky="nsew")

**def** go\_back(self):

*# Going back to the last accessed frame.*

**if** **not** self.\_stack.isEmpty(): *# if the stack is not empty*

previous\_frame\_class = self.\_stack.pop()

self.switch\_frame(previous\_frame\_class, going\_back=True)

**else**:

messagebox.showinfo("Back Navigation", "No previous screen to go back to.")

**def** load\_and\_scale\_image(pieceset, colour, piece\_type, size):

*# This function is responsible for loading an image and then scaling it to a smaller size.*

image = pygame.image.load(f'Assets/{pieceset}/{piece\_type}\_{colour}.png').convert\_alpha()

**return** pygame.transform.scale(image, size)

**def** initialise\_images(pieceset):

*# This function is responsible for initialising all of the images based off the user's requirements and selection of the pieceset.*

sizes = {

'3D': {'queen': (120, 120), 'king': (120, 120), 'rook': (120, 120), 'bishop': (120, 120),

'knight': (120, 120), 'pawn': (100, 100), 'small': (45, 45)},

'basic': {'queen': (90, 90), 'king': (90, 90), 'rook': (90, 90), 'bishop': (90, 90),

'knight': (90, 90), 'pawn': (80, 80), 'small': (45, 45)},

'anarcandy': {'queen': (80, 80), 'king': (80, 80), 'rook': (80, 80), 'bishop': (60, 80),

'knight': (70, 70), 'pawn': (70, 70), 'small': (30, 30)},

'merida': {'queen': (90, 90), 'king': (90, 90), 'rook': (90, 90), 'bishop': (90, 90),

'knight': (90, 90), 'pawn': (80, 80), 'small': (45, 45)}

}

warnings.filterwarnings("ignore", category=RuntimeWarning)

white\_pawn = load\_and\_scale\_image(pieceset, 'white', 'pawn', sizes[pieceset]['pawn'])

white\_pawn\_small = pygame.transform.scale(white\_pawn, sizes[pieceset]['small'])

black\_pawn = load\_and\_scale\_image(pieceset, 'black', 'pawn', sizes[pieceset]['pawn'])

black\_pawn\_small = pygame.transform.scale(black\_pawn, sizes[pieceset]['small'])

white\_knight = load\_and\_scale\_image(pieceset, 'white', 'knight', sizes[pieceset]['knight'])

white\_knight\_small = pygame.transform.scale(white\_knight, sizes[pieceset]['small'])

black\_knight = load\_and\_scale\_image(pieceset, 'black', 'knight', sizes[pieceset]['knight'])

black\_knight\_small = pygame.transform.scale(black\_knight, sizes[pieceset]['small'])

white\_bishop = load\_and\_scale\_image(pieceset, 'white', 'bishop', sizes[pieceset]['bishop'])

white\_bishop\_small = pygame.transform.scale(white\_bishop, sizes[pieceset]['small'])

black\_bishop = load\_and\_scale\_image(pieceset, 'black', 'bishop', sizes[pieceset]['bishop'])

black\_bishop\_small = pygame.transform.scale(black\_bishop, sizes[pieceset]['small'])

white\_rook = load\_and\_scale\_image(pieceset, 'white', 'rook', sizes[pieceset]['rook'])

white\_rook\_small = pygame.transform.scale(white\_rook, sizes[pieceset]['small'])

black\_rook = load\_and\_scale\_image(pieceset, 'black', 'rook', sizes[pieceset]['rook'])

black\_rook\_small = pygame.transform.scale(black\_rook, sizes[pieceset]['small'])

white\_queen = load\_and\_scale\_image(pieceset, 'white', 'queen', sizes[pieceset]['queen'])

white\_queen\_small = pygame.transform.scale(white\_queen, sizes[pieceset]['small'])

black\_queen = load\_and\_scale\_image(pieceset, 'black', 'queen', sizes[pieceset]['queen'])

black\_queen\_small = pygame.transform.scale(black\_queen, sizes[pieceset]['small'])

white\_king = load\_and\_scale\_image(pieceset, 'white', 'king', sizes[pieceset]['king'])

white\_king\_small = pygame.transform.scale(white\_king, sizes[pieceset]['small'])

black\_king = load\_and\_scale\_image(pieceset, 'black', 'king', sizes[pieceset]['king'])

black\_king\_small = pygame.transform.scale(black\_king, sizes[pieceset]['small'])

**return** white\_pawn, white\_pawn\_small, black\_pawn, black\_pawn\_small, white\_knight, white\_knight\_small, black\_knight, black\_knight\_small, white\_bishop, white\_bishop\_small, black\_bishop, black\_bishop\_small, white\_rook, white\_rook\_small, black\_rook, black\_rook\_small, white\_queen, white\_queen\_small, black\_queen, black\_queen\_small, white\_king, white\_king\_small, black\_king, black\_king\_small

**def** initialise\_game\_variables():

*# This function is responsible for setting and resetting all of the game variables. This is called at the start of the game and at the end of the game when the user wants to play again.*

white\_pieces = ['rook', 'knight', 'bishop', 'king', 'queen', 'bishop', 'knight', 'rook',

'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn']

white\_locations = [(0, 0), (1, 0), (2, 0), (3, 0), (4, 0), (5, 0), (6, 0), (7, 0),

(0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1)]

black\_pieces = ['rook', 'knight', 'bishop', 'king', 'queen', 'bishop', 'knight', 'rook',

'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn']

black\_locations = [(0, 7), (1, 7), (2, 7), (3, 7), (4, 7), (5, 7), (6, 7), (7, 7),

(0, 6), (1, 6), (2, 6), (3, 6), (4, 6), (5, 6), (6, 6), (7, 6)]

*# 0 - whites turn no selection: 1-whites turn piece selected: 2- black turn no selection, 3 - black turn piece selected*

turn\_step = 0

selection = 100

valid\_moves = []

white\_images = [white\_pawn, white\_queen, white\_king, white\_knight, white\_rook, white\_bishop]

white\_promotions = ['bishop', 'knight', 'rook', 'queen']

white\_moved = [False, False, False, False, False, False, False, False,

False, False, False, False, False, False, False, False]

small\_white\_images = [white\_pawn\_small, white\_queen\_small, white\_king\_small, white\_knight\_small,

white\_rook\_small, white\_bishop\_small]

black\_images = [black\_pawn, black\_queen, black\_king, black\_knight, black\_rook, black\_bishop]

small\_black\_images = [black\_pawn\_small, black\_queen\_small, black\_king\_small, black\_knight\_small,

black\_rook\_small, black\_bishop\_small]

black\_promotions = ['bishop', 'knight', 'rook', 'queen']

black\_moved = [False, False, False, False, False, False, False, False,

False, False, False, False, False, False, False, False]

piece\_list = ['pawn', 'queen', 'king', 'knight', 'rook', 'bishop']

*# check variables/ flashing counter*

counter = 0

winner = ''

game\_over = False

white\_ep = (100, 100)

black\_ep = (100, 100)

white\_promote = False

black\_promote = False

promo\_index = 100

check = False

castling\_moves = []

**return** white\_pieces, white\_locations, black\_pieces, black\_locations, turn\_step, selection, valid\_moves, white\_images, white\_promotions, white\_moved, small\_white\_images, black\_images, small\_black\_images, black\_promotions, black\_moved, piece\_list, counter, winner, game\_over, white\_ep, black\_ep, white\_promote, black\_promote, promo\_index, check, castling\_moves

**if** \_\_name\_\_ == "\_\_main\_\_":

app = App()

app.mainloop()

return\_value = app.return\_value *# The return value from clicking one of the buttons on the MainMenuScreen.*

*# Handling the return value*

**if** return\_value == 1: *# If the Standard Chess Button was clicked*

game\_run = False

pygame.init()

background = getBgCol() *# Get the background colour that the user wants*

sqcols = getSquareColour() *# Get the square colours that the user wants*

pieceset = selectPieceSet() *# Get the pieceset that the user wants*

game\_run = True

**if** game\_run:

WIDTH = 1000

HEIGHT = 900

screen = pygame.display.set\_mode([WIDTH, HEIGHT])

pygame.display.set\_caption('Two-Player Pygame Chess!')

font = pygame.font.Font('freesansbold.ttf', 20)

medium\_font = pygame.font.Font('freesansbold.ttf', 40)

big\_font = pygame.font.Font('freesansbold.ttf', 50)

timer = pygame.time.Clock()

fps = 60

white\_pawn, white\_pawn\_small, black\_pawn, black\_pawn\_small, white\_knight, white\_knight\_small, black\_knight, black\_knight\_small, white\_bishop, white\_bishop\_small, black\_bishop, black\_bishop\_small, white\_rook, white\_rook\_small, black\_rook, black\_rook\_small, white\_queen, white\_queen\_small, black\_queen, black\_queen\_small, white\_king, white\_king\_small, black\_king, black\_king\_small =initialise\_images(pieceset) *# Initialise the images required to make the display*

white\_pieces, white\_locations, black\_pieces, black\_locations, turn\_step, selection, valid\_moves, white\_images, white\_promotions, white\_moved, small\_white\_images, black\_images, small\_black\_images, black\_promotions, black\_moved, piece\_list, counter, winner, game\_over, white\_ep, black\_ep, white\_promote, black\_promote, promo\_index, check, castling\_moves =initialise\_game\_variables() *# Initialise the game variables*

black\_options, castling\_moves = check\_options(black\_pieces, black\_locations, 'black', castling\_moves, check) *# Get black's legal moves.*

white\_options, castling\_moves = check\_options(white\_pieces, white\_locations, 'white', castling\_moves, check) *# Get white's legal moves.*

run = True

**while** run:

timer.tick(fps)

**if** counter < 30:

counter += 1

**else**:

counter = 0

screen.fill(background)

draw\_board(sqcols[0], sqcols[1])

draw\_pieces(pieceset)

draw\_check(turn\_step, counter)

**if** **not** game\_over:

white\_promote, black\_promote, promo\_index = check\_promotion()

**if** white\_promote **or** black\_promote:

draw\_promotion()

check\_promo\_select()

**if** selection != 100:

valid\_moves = check\_valid\_moves(white\_options, black\_options, turn\_step, selection)

draw\_valid(valid\_moves, turn\_step)

**if** selected\_piece == 'king':

draw\_castling(turn\_step, castling\_moves)

*# event handling*

**for** event **in** pygame.event.get():

**if** event.type == pygame.QUIT:

run = False

**if** event.type == pygame.MOUSEBUTTONDOWN **and** event.button == 1 **and** **not** game\_over:

x\_coord = event.pos[0] // 100 *# Get the x position of the mouse click*

y\_coord = event.pos[1] // 100 *# Get the y position of the mouse click*

click\_coords = (x\_coord, y\_coord)

**if** turn\_step <= 1: *# If it's white's turn*

**if** click\_coords == (8, 8) **or** click\_coords == (9, 8): *# If the resign button is clicked*

winner = 'black' *# Black wins*

**elif** len(white\_options) == 0: *# We have a stalemate*

winner = 'draw'

**if** click\_coords **in** white\_locations: *# Check if a piece has been selected*

selection = white\_locations.index(click\_coords)

selected\_piece = white\_pieces[selection]

**if** turn\_step == 0:

turn\_step = 1 *# Move onto the next game phase where you have to select a destination for that piece*

**if** click\_coords **in** valid\_moves **and** selection != 100:

white\_ep = check\_en\_passant(turn\_step, white\_locations[selection], click\_coords)

white\_locations[selection] = click\_coords

white\_moved[selection] = True

**if** click\_coords **in** black\_locations: *# if the second set of coordinates is in the black\_locations (i.e. where the black pieces are)*

black\_piece = black\_locations.index(click\_coords) *# get the index of the piece that is being attacked and clicked on*

**if** black\_pieces[black\_piece] == 'king': *# If the piece is the king,*

winner = 'white' *# White wins.*

black\_pieces.pop(black\_piece) *# remove the piece from the board*

black\_locations.pop(black\_piece) *# remove the piece from the list of black locations*

black\_moved.pop(black\_piece) *# remove the piece from the list of pieces that can be moved.*

**if** click\_coords == black\_ep: *# If it's an en passant move, remove the piece that has been captured by en passant*

black\_piece = black\_locations.index((black\_ep[0], black\_ep[1] - 1))

black\_pieces.pop(black\_piece)

black\_locations.pop(black\_piece)

black\_moved.pop(black\_piece)

black\_options, castling\_moves = check\_options(black\_pieces, black\_locations, 'black', castling\_moves, check) *# Get the new set of legal mvoes*

white\_options, castling\_moves = check\_options(white\_pieces, white\_locations, 'white', castling\_moves, check) *# Get the new set of legal moves*

turn\_step = 2 *# Black's turn*

selection = 100

valid\_moves = []

*# add option to castle*

**elif** selection != 100 **and** selected\_piece == 'king': *# Check whether castling is legal*

**for** q **in** range(len(castling\_moves)):

**if** click\_coords == castling\_moves[q][0]:

white\_locations[selection] = click\_coords

white\_moved[selection] = True

**if** click\_coords == (1, 0):

rook\_coords = (0, 0)

**else**:

rook\_coords = (7, 0)

rook\_index = white\_locations.index(rook\_coords)

white\_locations[rook\_index] = castling\_moves[q][1]

black\_options, castling\_moves = check\_options(black\_pieces, black\_locations, 'black', castling\_moves, check)

white\_options, castling\_moves = check\_options(white\_pieces, white\_locations, 'white', castling\_moves, check)

turn\_step = 2

selection = 100

valid\_moves = []

**if** turn\_step > 1: *# Black's turn*

**if** click\_coords == (8, 8) **or** click\_coords == (9, 8): *# If the resign button is pressed*

winner = 'white' *# White wins.*

**elif** len(black\_options) == 0: *# Stalemate*

winner = 'draw'

**if** click\_coords **in** black\_locations:

selection = black\_locations.index(click\_coords)

*# check what piece is selected, so you can only draw castling moves if king is selected*

selected\_piece = black\_pieces[selection]

**if** turn\_step == 2:

turn\_step = 3 *# Time to move said selected piece to a destination.*

**if** click\_coords **in** valid\_moves **and** selection != 100: *# If the move is legal, make the move.*

black\_ep = check\_en\_passant(turn\_step, black\_locations[selection], click\_coords)

black\_locations[selection] = click\_coords

black\_moved[selection] = True

**if** click\_coords **in** white\_locations: *# If the destination has a white piece on it already.*

white\_piece = white\_locations.index(click\_coords) *# Works out what is the piece on the square.*

**if** white\_pieces[white\_piece] == 'king': *# If the piece is the king, the king is captured and black wins.*

winner = 'black' *# Black wins*

white\_pieces.pop(white\_piece)

white\_locations.pop(white\_piece) *# Remove the captured piece from the board and we no longer need to keep a track of its location due to it not being present on the board.*

white\_moved.pop(white\_piece)

**if** click\_coords == white\_ep: *# Make the en passant capture if it is an en passant capture.*

white\_piece = white\_locations.index((white\_ep[0], white\_ep[1] + 1))

white\_pieces.pop(white\_piece)

white\_locations.pop(white\_piece)

white\_moved.pop(white\_piece)

black\_options, castling\_moves = check\_options(black\_pieces, black\_locations, 'black', castling\_moves, check) *# Switch back to white's turn*

white\_options, castling\_moves = check\_options(white\_pieces, white\_locations, 'white', castling\_moves, check)

turn\_step = 0

selection = 100

valid\_moves = []

*# add option to castle*

**elif** selection != 100 **and** selected\_piece == 'king':

**for** q **in** range(len(castling\_moves)):

**if** click\_coords == castling\_moves[q][0]:

black\_locations[selection] = click\_coords

black\_moved[selection] = True

**if** click\_coords == (1, 7):

rook\_coords = (0, 7)

**else**:

rook\_coords = (7, 7)

rook\_index = black\_locations.index(rook\_coords)

black\_locations[rook\_index] = castling\_moves[q][1]

black\_options, castling\_moves = check\_options(black\_pieces, black\_locations, 'black', castling\_moves, check)

white\_options, castling\_moves = check\_options(white\_pieces, white\_locations, 'white', castling\_moves, check)

turn\_step = 0

selection = 100

valid\_moves = []

**if** event.type == pygame.KEYDOWN **and** game\_over: *# If the game is over and a key is pressed down,*

**if** event.key == pygame.K\_RETURN: *# If said key is the ENTER key,*

white\_pieces, white\_locations, black\_pieces, black\_locations, turn\_step, selection, valid\_moves, white\_images, white\_promotions, white\_moved, small\_white\_images, black\_images, small\_black\_images, black\_promotions, black\_moved, piece\_list, counter, winner, game\_over, white\_ep, black\_ep, white\_promote, black\_promote, promo\_index, check, castling\_moves =initialise\_game\_variables() *# reinitialise the game variables*

winner = '' *# set the winner to '' as there is no winner and we have a new game on our hands*

black\_options, castling\_moves = check\_options(black\_pieces, black\_locations, 'black', castling\_moves, check) *# Get the legal moves of the position*

white\_options, castling\_moves = check\_options(white\_pieces, white\_locations, 'white', castling\_moves, check)

**if** winner != '': *# If the king has been captured at any point, then we have a winner according to the code logic.*

game\_over = True *# flag variable to indicate that the game is over*

draw\_game\_over(winner) *# Draw the game over message to indicate the outcome of the game.*

pygame.display.update()

pygame.quit()

**elif** return\_value == 2:

running = True

rules = False

**print**("""\_\_\_\_\_\_ \_ \_\_\_\_\_\_ \_ \_ \_ \_

| \_\_\_ (\_) | \_\_\_ **\**  | | | | (\_) | |

| |\_/ /\_ \_\_ \_ | |\_/ / \_\_ \_ \_ \_\_ \_\_ \_ | | | | \_\_ \_ \_ \_\_ \_ \_\_ \_ \_ \_\_ | |\_

| \_\_\_ **\** |/ \_` | | \_\_\_ **\/** \_` | '\_ **\** / \_` | | | | |/ \_` | '\_\_| |/ \_` | '\_ **\|** \_\_|

| |\_/ / | (\_| | | |\_/ / (\_| | | | | (\_| | **\ \\_**/ / (\_| | | | | (\_| | | | | |\_

**\\_**\_\_\_/|\_|**\\_**\_, | **\\_**\_\_\_/ **\\_**\_,\_|\_| |\_|**\\_**\_, | **\\_**\_\_/ **\\_**\_,\_|\_| |\_|**\\_**\_,\_|\_| |\_|**\\_**\_|

\_\_/ / \_\_/ /

|\_\_\_/ |\_\_\_/ """)

**print**("**\n\n**")

**print**("Welcome to my variant.")

time.sleep(2)

**print**("Would you like to learn the rules?")

**while** rules == False:

y\_or\_n = input(">(Y/N)").lower()

**if** y\_or\_n == "y":

**print**("1) It is the first of its kind and the rules are simple.")

**print**("**\n**")

time.sleep(3)

**print**("2) All of the pieces move like normal chess but there are some new rules.")

**print**("**\n**")

time.sleep(3)

**print**("3) There is no castling, no checkmate, no en passant and no check, which means that you will have to rely on your brain power and wit to capture all of the pieces.")

**print**("**\n**")

time.sleep(4)

**print**("4) There are an additional two pieces to this game: the Grand Empress and the Serpent.")

**print**("**\n**")

time.sleep(3)

**print**("5) The Grand Empress moves similar to a knight but can't capture pieces. It is a healer against the serpent's poison.")

**print**("**\n**")

time.sleep(8)

**print**("6) The serpent can move forward four squares at a time but can't move in any other direction.")

**print**("**\n**")

time.sleep(5)

**print**("You have two serpent pieces: one is able to capture pieces while the other isn't able to capture them as it is not poisonous.")

**print**("**\n**")

time.sleep(5)

**print**("If the Grand Empress is not on a neighbouring square to the piece that is being captured by the serpent, then the piece will be removed from the board.")

**print**("**\n**")

time.sleep(6)

**print**("Once healed, both the serpent and the Grand Empress die and are removed from the game.")

**print**("**\n**")

time.sleep(3)

rules = True

**elif** y\_or\_n == "n":

rules = True

**else**:

**print**("Please only enter y or n.")

bgcol = getBgCol()

colours= getSquareColour()

pieceset = selectPieceSet()

run = True

**if** run == True:

a = BBApp()

game = a.main(bgcol, colours, pieceset)

warnings.filterwarnings("ignore", category=RuntimeWarning)

### **ViewLichessStats.py**

**import** json

**import** berserk

**import** matplotlib.pyplot **as** plt

**from** datetime **import** datetime

**def** return\_lichess\_data():

API\_TOKEN = "lip\_dWBg5ZzgY80s4F3rbE9y" *#* *This is the API token that is linked to my end user's account. I was able to obtain this through detailing a series of instructions on an email.*

session = berserk.TokenSession(API\_TOKEN)

client = berserk.Client(session=session) *# There is no way of me being able to access my end user's details other than through using the Python client for the Lichess API.*

**try**:

data = client.account.get() *# This function gets all of the data that is associated with the account that the API token is linked to.*

**except** berserk.BerserkError **as** e: *# To deal with any errors that may arise.*

**print**(f"Error retrieving lichess data: {e}")

data = None

**if** data:

*# Save data to a JSON file*

**with** open('player\_data.json', 'w') **as** file:  *# The data is arriving in a dictionary format so the data is being stored in a JSON file before being manipulated.*

json.dump(data, file, default=str) *# Using default=str to handle datetime objects*

**def** fetch\_games(user\_id, variant):

*# This function returns a list of the last 10 games of a particular variant in PGN format*

API\_TOKEN = "lip\_dWBg5ZzgY80s4F3rbE9y"

session = berserk.TokenSession(API\_TOKEN)

client = berserk.Client(session=session)

**try**:

*# Fetch the last 10 games played by the user in PGN format*

games\_pgn = client.games.export\_by\_player(user\_id, rated=True, perf\_type=variant, clocks=False, max=10, as\_pgn=True)

**return** games\_pgn

**except** berserk.BerserkError **as** e:

**print**(f"Error fetching games: {e}")

**return** None

**def** display\_pgn(games\_pgn):

*# Print PGN for each game*

**for** game\_pgn **in** games\_pgn:

**print**(game\_pgn)

**print**("**\n\n**")

**def** draw\_ratings\_graphs(variants\_data):

*# Plotting the line graph for all variants*

plt.figure(figsize=(10, 6))

**for** variant, (dates, ratings) **in** variants\_data.items(): *#The key*

plt.plot(dates, ratings, marker='o', label=f'{variant.capitalize()} Ratings')

plt.xlabel('Date')

plt.ylabel('Rating')

plt.title("VishnuVadlamani's lichess ratings")

plt.legend()

plt.xticks(rotation=45, ha='right')

plt.tight\_layout()

plt.savefig('all\_variants\_line\_graph.png') *# Save the graph as an image*

plt.gcf().canvas.manager.set\_window\_title("VishnuVadlamani's lichess ratings")

plt.show()

*# Empty the "player\_data.json" file*

**with** open('player\_data.json', 'w') **as** empty\_file: *# The json file is emptied so there is no confusion with the data the next instance that the code is executed (i.e. the next time that the user enters 2).*

empty\_file.write("")

**def** get\_choice():

*# This function gets the choice from the user.*

**while** True:

**try**:

c = input("--->").lower() *# Get the user choice of what they want to do.*

**if** c == "q" **or** 1 <= int(c) <= 2:

**return** c *# A valid input has been provided so it is returned.*

**else**:

**raise** ValueError("Invalid choice. Please enter 1 or 2 or enter Q to quit.") *# rogue input*

**except** ValueError:

**print**("Rogue input detected. Please enter a valid choice.")

**def** print\_choices():

*# This function output to the user the available choices.*

**print**("1-View PGN of the last 10 games played")

**print**("2-View a graph of lichess ratings over time")

**print**("Q - Quit the program")

**def** print\_choices():

*# This function output to the user the available choices.*

**print**("1-View PGN of the last 10 games played")

**print**("2-View a graph of lichess ratings over time")

**print**("Q - Quit the program")

**def** get\_variant():

*# This function asks the user to select a variant that they would like to see the last 10 games of.*

valid\_variant = False

**while** **not** valid\_variant:

variant = input("Select a variant (Bullet, Blitz, Rapid):").lower()

**if** variant **not** **in** ["bullet", "blitz", "rapid"] **and** variant != "": *# My end user has only played four classical games to date which means that it isn't possible to fetch the game data for the last 10 games. Hence, it hasn't been included in the variant list.*

**print**("Please select a valid variant or press enter to exit")

**else**:

valid\_variant = True

**return** variant *# When a valid variant is selected, the variant will be returned*

**def** get\_rating\_data():

*# From the data stored in the json file after the data has been fetched from the API, the data is manipulated to fetch the ratings.*

*# Each rating is then stored with the date that the rating is fetched in a corresponding text file based off the variant name*

*# Load data from the JSON file*

**with** open('player\_data.json', 'r') **as** file:

data = json.load(file)

*# Extracting details*

perfs = data['perfs']

*# Dictionary to store dates and ratings for each variant*

variants\_data = {}

*# Extract ratings for each game variant*

**for** variant **in** ['bullet', 'blitz', 'rapid', 'classical']:

rating = perfs.get(variant, {}).get('rating', None)

**if** rating **is** **not** None:

*# Save rating to the text file*

file\_path = f'{variant}\_ratings.txt'

current\_entry = f"{datetime.now().strftime('%d/%m/%Y')}**\n**{rating}**\n**"

*# Open the file in append mode and create it if it doesn't exist*

**with** open(file\_path, 'a+') **as** rating\_file:

*# Move the cursor to the beginning of the file to read existing entries*

rating\_file.seek(0)

existing\_entries = rating\_file.readlines()

**if** current\_entry **not** **in** existing\_entries:

rating\_file.write(current\_entry)

*# Append dates and ratings to the dictionary*

dates, ratings = [], []

**with** open(file\_path, 'r') **as** variant\_file:

lines = variant\_file.readlines()

**for** i **in** range(0, len(lines), 2):

date\_str, rating\_str = lines[i].strip(), lines[i+1].strip()

*# Check if date\_str is not an empty string before converting*

**if** date\_str:

dates.append(datetime.strptime(date\_str, "%d/%m/%Y"))

ratings.append(int(rating\_str))

variants\_data[variant] = (dates, ratings)

**return** variants\_data

**def** main():

*# The main function.*

run = True

**while** run: *#this will run indefinitely until the user has entered 'q' in the input field.*

print\_choices()

user\_choice = get\_choice()

*# user choice handling*

**if** user\_choice == "1":

variant = get\_variant()

user\_id = "VishnuVadlamani" *# my end user's account username*

games\_pgn = fetch\_games(user\_id, variant)

**if** games\_pgn:

display\_pgn(games\_pgn)

**elif** user\_choice == "2":

return\_lichess\_data() *# This function retrieves the Lichess data and then stores in the player\_data.json file.*

rating\_data = get\_rating\_data() *# This splits the data by rating into different text files.*

draw\_ratings\_graphs(rating\_data) *# This graphically represents the rating data that is present in the text files.*

**elif** user\_choice.lower() == 'q': *# If the user wants to quit*

run = False

exit()

**if** \_\_name\_\_ == "\_\_main\_\_":

main()

### **bigbangchessapp.py**

**import** pygame

**import** warnings

**from** pawnclass **import** Pawn

**from** bishopclass **import** Bishop

**from** serpentclass **import** Serpent

**from** kingclass **import** King

**from** queenclass **import** Queen

**from** empressclass **import** GrandEmpress

**from** boardclass **import** Board

**class** BBApp(object):

**def** \_\_init\_\_(self):

pygame.init()

self.\_\_screen = pygame.display.set\_mode((1000, 900))

pygame.display.set\_caption("Chess Game")

self.\_\_board = Board()

**def** main(self, bgcol, colours, pieceset):

*# main function*

pygame.init()

pygame.display.set\_caption('Two-Player Pygame Chess!')

timer = pygame.time.Clock()

fps = 60

*# 0 - whites turn no selection: 1-whites turn piece selected: 2- black turn no selection, 3 - black turn piece selected*

valid\_moves = []

warnings.filterwarnings("ignore", category=RuntimeWarning)

black\_queen = pygame.image.load('Assets/'+str(pieceset)+'/queen\_black.png').convert\_alpha()

black\_king = pygame.image.load('Assets/'+str(pieceset)+'/king\_black.png').convert\_alpha()

black\_empress = pygame.image.load('Assets/GrandEmpressBlack.png').convert\_alpha()

black\_bishop = pygame.image.load('Assets/'+str(pieceset)+'/bishop\_black.png').convert\_alpha()

black\_serpent = pygame.image.load('Assets/BlackSerpent.png').convert\_alpha()

black\_pawn = pygame.image.load('Assets/'+str(pieceset)+'/pawn\_black.png').convert\_alpha()

warnings.filterwarnings("ignore", category=RuntimeWarning)

white\_queen = pygame.image.load('Assets/'+str(pieceset)+'/queen\_white.png').convert\_alpha()

white\_king = pygame.image.load('Assets/'+str(pieceset)+'/king\_white.png').convert\_alpha()

white\_empress = pygame.image.load('Assets/GrandEmpressWhite.png').convert\_alpha()

white\_bishop = pygame.image.load('Assets/'+str(pieceset)+'/bishop\_white.png').convert\_alpha()

white\_serpent = pygame.image.load('Assets/WhiteSerpent.png').convert\_alpha()

white\_pawn = pygame.image.load('Assets/'+str(pieceset)+'/pawn\_white.png').convert\_alpha()

warnings.filterwarnings("ignore", category=RuntimeWarning)

self.\_\_board.\_setImages(black\_queen, black\_king, black\_empress, black\_bishop, black\_serpent, black\_pawn, white\_queen, white\_king, white\_empress, white\_bishop, white\_serpent, white\_pawn)

game\_over = False

*# main game loop*

black\_options, black\_poison\_options = self.\_\_board.\_check\_options(self.\_\_board.\_black\_pieces, self.\_\_board.\_black\_locations, 'black') *# Fetches the legal moves for black*

white\_options, white\_poison\_options = self.\_\_board.\_check\_options(self.\_\_board.\_white\_pieces, self.\_\_board.\_white\_locations, 'white') *# Fetches the legal moves for white*

run = True *# Start of the game loop. Setting this to False will end the game loop*

serpent\_selected = False

**while** run:

timer.tick(fps)

**if** self.\_\_board.\_counter < 30:

self.\_\_board.\_counter += 1

**else**:

self.\_\_board.\_counter = 0

self.\_\_screen.fill(bgcol) *# fill the screen*

self.\_\_board.\_draw\_board(self.\_\_screen, colours[0], colours[1]) *# draw the board*

self.\_\_board.\_draw\_pieces(self.\_\_screen, pieceset) *# draw the pieces*

**if** **not** game\_over:

self.\_\_board.\_white\_promote, self.\_\_board.\_black\_promote, self.\_\_board.\_promo\_index = self.\_\_board.\_check\_promotion() *#Check if a pawn has reach the final ranks for black or white at the start of each loop.*

**if** self.\_\_board.\_white\_promote **or** self.\_\_board.\_black\_promote: *# If there is a pawn that has reach the final ranks*

self.\_\_board.\_draw\_promotion(self.\_\_screen) *# Draw the promotion*

self.\_\_board.\_check\_promo\_select() *# Select the piece that the pawn should promote to.*

**if** self.\_\_board.\_selection != 100: *# If the game is not over*

valid\_moves = self.\_\_board.\_check\_valid\_moves(white\_options, black\_options) *# Check all of the legal moves*

self.\_\_board.\_draw\_valid(self.\_\_screen, valid\_moves) *# Draw the valid moves that are returned from the list.*

*# event handling*

**for** event **in** pygame.event.get():

**if** event.type == pygame.QUIT:

run = False *# Cross in the top right corner of the screen has been clicked indicating that the user no longer wants to play.*

**if** event.type == pygame.MOUSEBUTTONDOWN **and** event.button == 1 **and** **not** game\_over: *# Checks for the piece selected.*

x\_coord = event.pos[0] // 100 *# x coordinate of mouse press*

y\_coord = event.pos[1] // 100 *# y coordinate of mouse press*

click\_coords = (x\_coord, y\_coord) *# coordinates of mouse click*

**if** self.\_\_board.\_turn\_step <= 1: *# White's turn*

**if** click\_coords **in** self.\_\_board.\_white\_locations: *#If a white piece has been selected.*

self.\_\_board.\_selection = self.\_\_board.\_white\_locations.index(click\_coords) *#Find the piece.*

**if** self.\_\_board.\_turn\_step == 0:

self.\_\_board.\_turn\_step = 1 *# Change state from select piece to select location for the piece.*

**if** click\_coords **in** valid\_moves **and** self.\_\_board.\_selection != 100: *# If the move is legal*

self.\_\_board.\_white\_locations[self.\_\_board.\_selection] = click\_coords *# Make the move.*

**if** click\_coords **in** self.\_\_board.\_black\_locations: *# If the second mouse click is legal and the square it is being moved to has a black piece on it.*

white\_piece = self.\_\_board.\_white\_locations.index(click\_coords) *# work out what the white piece is*

**if** self.\_\_board.\_white\_pieces[white\_piece] == "serpent": *# has the serpent piece been selected?*

serpent\_selected = True *# flag variable to indicate that the serpent has been selected.*

**if** serpent\_selected == True:

**if** click\_coords **in** white\_poison\_options: *# There is an enemy piece on*

neighbouring\_squares = []

*# All of these different shifts are from black's perspective. If it is from white's perspective, it would be the complete opposite as I have chosen to implement the board from the black perspective*

*# So you are looking at the board from the perspective of the person playing with the black pieces.*

**if** 0 <= click\_coords[0]+1 <=7 **and** 0 <= click\_coords[1] <= 7 **and** (click\_coords[0]+1, click\_coords[1]) **in** self.\_\_board.\_black\_locations: *# One square to the right of click\_coords*

neighbouring\_squares.append((click\_coords[0]+1, click\_coords[1]))

**if** 0 <= click\_coords[0]-1 <=7 **and** 0 <= click\_coords[1] <= 7 **and** (click\_coords[0]-1, click\_coords[1]) **in** self.\_\_board.\_black\_locations: *# One square to the left of click\_coords*

neighbouring\_squares.append((click\_coords[0]-1, click\_coords[1]))

**if** 0 <= click\_coords[0] <=7 **and** 0 <= click\_coords[1]+1 <= 7 **and** (click\_coords[0],click\_coords[1]+1) **in** self.\_\_board.\_black\_locations: *# One square up from click\_coords*

neighbouring\_squares.append((click\_coords[0], click\_coords[1]+1))

**if** 0 <= click\_coords[0] <=7 **and** 0 <= click\_coords[1]-1 <= 7 **and** (click\_coords[0], click\_coords[1]-1) **in** self.\_\_board.\_black\_locations: *# One square down from click\_coords*

neighbouring\_squares.append((click\_coords[0]-1, click\_coords[1]))

**if** 0 <= click\_coords[0]+1 <=7 **and** 0 <= click\_coords[1]+1 <= 7 **and** (click\_coords[0]+1, click\_coords[1]+1) **in** self.\_\_board.\_black\_locations : *#Diagonal top right from click\_coords*

neighbouring\_squares.append((click\_coords[0]+1, click\_coords[1]+1))

**if** 0 <= click\_coords[0]+1 <=7 **and** 0 <= click\_coords[1]-1 <= 7 **and** (click\_coords[0]+1, click\_coords[1]-1) **in** self.\_\_board.\_black\_locations: *# Diagonal bottom right from click\_coords*

neighbouring\_squares.append((click\_coords[0]+1, click\_coords[1]-1))

**if** 0 <= click\_coords[0]-1 <=7 **and** 0 <= click\_coords[1]+1 <= 7 **and** (click\_coords[0]-1, click\_coords[1]+1) **in** self.\_\_board.\_black\_locations: *# Diagonal top left from click\_coords*

neighbouring\_squares.append((click\_coords[0]+1, click\_coords[1]))

**if** 0 <= click\_coords[0]-1 <=7 **and** 0 <= click\_coords[1]-1 <= 7 **and** (click\_coords[0]-1, click\_coords[1]-1) **in** self.\_\_board.\_black\_locations: *#Diagonal bottom left from click\_coords*

neighbouring\_squares.append((click\_coords[0]-1, click\_coords[1]))

empress\_nearby = False

**for** square **in** neighbouring\_squares:

**if** square **in** self.\_\_board.\_black\_locations:

piece\_index = self.\_\_board.\_black\_locations.index(square)

**if** self.\_\_board.\_black\_pieces[piece\_index] == "empress":

empress\_nearby = True

**break**

**if** empress\_nearby:

*# Remove white serpent and empress, update lists*

white\_piece = self.\_\_board.\_white\_locations.index(self.\_\_board.\_white\_locations[self.\_\_board.\_selection])

empress\_index = self.\_\_board.\_black\_locations.index(square)

self.\_\_board.\_white\_pieces.pop(white\_piece)

self.\_\_board.\_white\_locations.pop(white\_piece)

self.\_\_board.\_black\_pieces.pop(empress\_index)

self.\_\_board.\_black\_locations.pop(empress\_index)

**else**:

self.\_\_board.\_black\_pieces.pop(black\_piece) *# remove the piece from the board*

self.\_\_board.\_black\_locations.pop(black\_piece) *# Remove the location that the piece was on from the list of locations that black currently has pieces on.*

**else**:

black\_piece = self.\_\_board.\_black\_locations.index(click\_coords)

**if** self.\_\_board.\_black\_pieces[black\_piece] == 'king': *# If the piece is the king,*

piece\_captured = King()

self.\_\_board.\_white\_material\_count += piece\_captured.\_value

**elif** self.\_\_board.\_black\_pieces[black\_piece] == 'queen': *# If the piece is a queen,*

piece\_captured = Queen()

self.\_\_board.\_white\_material\_count += piece\_captured.\_value

**elif** self.\_\_board.\_black\_pieces[black\_piece] == 'bishop': *# If the piece is a queen,*

piece\_captured = Bishop()

self.\_\_board.\_white\_material\_count += piece\_captured.\_value

**elif** self.\_\_board.\_black\_pieces[black\_piece] == 'serpent': *# If the piece is a queen,*

piece\_captured = Serpent()

self.\_\_board.\_white\_material\_count += piece\_captured.\_value

**elif** self.\_\_board.\_black\_pieces[black\_piece] == 'empress': *# If the piece is a queen,*

piece\_captured = GrandEmpress()

self.\_\_board.\_white\_material\_count += piece\_captured.\_value

**else**:

piece\_captured = Pawn()

self.\_\_board.\_white\_material\_count += piece\_captured.\_value

self.\_\_board.\_black\_pieces.pop(black\_piece) *# remove the piece from the board*

self.\_\_board.\_black\_locations.pop(black\_piece) *# remove the piece from the list of black locations*

*# Check the next set of legal moves in the new position.*

black\_options, black\_poison\_options = self.\_\_board.\_check\_options(self.\_\_board.\_black\_pieces, self.\_\_board.\_black\_locations, 'black')

white\_options , white\_poison\_options= self.\_\_board.\_check\_options(self.\_\_board.\_white\_pieces, self.\_\_board.\_white\_locations, 'white')

self.\_\_board.\_turn\_step = 2 *# Black's turn now.*

self.\_\_board.\_selection = 100

valid\_moves = []

**if** self.\_\_board.\_turn\_step > 1:

**if** click\_coords **in** self.\_\_board.\_black\_locations:

self.\_\_board.\_selection = self.\_\_board.\_black\_locations.index(click\_coords) *#Find the piece.*

**if** self.\_\_board.\_turn\_step == 2:

self.\_\_board.\_turn\_step = 3

**if** click\_coords **in** valid\_moves **and** self.\_\_board.\_selection != 100: *# If the move is legal*

self.\_\_board.\_black\_locations[self.\_\_board.\_selection] = click\_coords *# Make the move.*

**if** click\_coords **in** self.\_\_board.\_white\_locations: *# If the second mouse click is legal and the square it is being moved to has a black piece on it.*

black\_piece = self.\_\_board.\_black\_locations.index(click\_coords)

**if** self.\_\_board.\_black\_pieces[black\_piece] == "serpent":

serpent\_selected = True

**if** serpent\_selected == True:

**if** click\_coords **in** black\_poison\_options:

neighbouring\_squares = []

**if** 0 <= click\_coords[0]+1 <=7 **and** 0 <= click\_coords[1] <= 7 **and** (click\_coords[0]+1, click\_coords[1]) **in** self.\_\_board.\_white\_locations: *# One square to the right of click\_coords*

neighbouring\_squares.append((click\_coords[0]+1, click\_coords[1]))

**if** 0 <= click\_coords[0]-1 <=7 **and** 0 <= click\_coords[1] <= 7 **and** (click\_coords[0]-1, click\_coords[1]) **in** self.\_\_board.\_white\_locations: *# One square to the left of click\_coords*

neighbouring\_squares.append((click\_coords[0]-1, click\_coords[1]))

**if** 0 <= click\_coords[0] <=7 **and** 0 <= click\_coords[1]+1 <= 7 **and** (click\_coords[0],click\_coords[1]+1) **in** self.\_\_board.\_white\_locations: *# One square up from click\_coords*

neighbouring\_squares.append((click\_coords[0], click\_coords[1]+1))

**if** 0 <= click\_coords[0] <=7 **and** 0 <= click\_coords[1]-1 <= 7 **and** (click\_coords[0], click\_coords[1]-1) **in** self.\_\_board.\_white\_locations: *# One square down from click\_coords*

neighbouring\_squares.append((click\_coords[0]-1, click\_coords[1]))

**if** 0 <= click\_coords[0]+1 <=7 **and** 0 <= click\_coords[1]+1 <= 7 **and** (click\_coords[0]+1, click\_coords[1]+1) **in** self.\_\_board.\_white\_locations : *#Diagonal top right from click\_coords*

neighbouring\_squares.append((click\_coords[0]+1, click\_coords[1]+1))

**if** 0 <= click\_coords[0]+1 <=7 **and** 0 <= click\_coords[1]-1 <= 7 **and** (click\_coords[0]+1, click\_coords[1]-1) **in** self.\_\_board.\_white\_locations: *# Diagonal bottom right from click\_coords*

neighbouring\_squares.append((click\_coords[0]+1, click\_coords[1]-1))

**if** 0 <= click\_coords[0]-1 <=7 **and** 0 <= click\_coords[1]+1 <= 7 **and** (click\_coords[0]-1, click\_coords[1]+1) **in** self.\_\_board.\_white\_locations: *# Diagonal top left from click\_coords*

neighbouring\_squares.append((click\_coords[0]+1, click\_coords[1]))

**if** 0 <= click\_coords[0]-1 <=7 **and** 0 <= click\_coords[1]-1 <= 7 **and** (click\_coords[0]-1, click\_coords[1]-1) **in** self.\_\_board.\_white\_locations: *#Diagonal bottom left from click\_coords*

neighbouring\_squares.append((click\_coords[0]-1, click\_coords[1]))

empress\_nearby = False

**for** square **in** neighbouring\_squares:

**if** square **in** self.\_\_board.\_white\_locations:

piece\_index = self.\_\_board.\_white\_locations.index(square)

**if** self.\_\_board.\_white\_pieces[piece\_index] == "empress":

empress\_nearby = True

**break**

**if** empress\_nearby:

*# Remove black serpent and empress, update lists*

black\_piece = self.\_\_board.\_black\_locations.index(self.\_\_board.\_black\_locations[self.\_\_board.\_selection])

empress\_index = self.\_\_board.\_white\_locations.index(square)

self.\_\_board.\_black\_pieces.pop(black\_piece)

self.\_\_board.\_black\_locations.pop(black\_piece)

self.\_\_board.\_white\_pieces.pop(empress\_index)

self.\_\_board.\_white\_locations.pop(empress\_index)

**else**:

self.\_\_board.\_white\_pieces.pop(white\_piece) *# remove the piece from the board*

self.\_\_board.\_white\_locations.pop(white\_piece) *# Remove the location that the piece was on from the list of locations that black currently has pieces on.*

**else**:

white\_piece = self.\_\_board.\_white\_locations.index(click\_coords)

**if** self.\_\_board.\_white\_pieces[white\_piece] == 'king': *# If the piece is the king,*

piece\_captured = King()

self.\_\_board.\_black\_material\_count += piece\_captured.\_value

**elif** self.\_\_board.\_white\_pieces[white\_piece] == 'queen': *# If the piece is a queen,*

piece\_captured = Queen()

self.\_\_board.\_black\_material\_count += piece\_captured.\_value

**elif** self.\_\_board.\_black\_pieces[black\_piece] == 'bishop': *# If the piece is a bishop,*

piece\_captured = Bishop()

self.\_\_board.\_black\_material\_count += piece\_captured.\_value

**elif** self.\_\_board.\_black\_pieces[black\_piece] == 'serpent': *# If the piece is a serpent,*

piece\_captured = Serpent()

self.\_\_board.\_black\_material\_count += piece\_captured.\_value

**elif** self.\_\_board.\_black\_pieces[black\_piece] == 'empress': *# If the piece is an empress,*

piece\_captured = GrandEmpress()

self.\_\_board.\_black\_material\_count += piece\_captured.\_value

**else**: *# The only other possible piece is a pawn*

piece\_captured = Pawn()

self.\_\_board.\_black\_material\_count += piece\_captured.\_value

self.\_\_board.\_white\_pieces.pop(white\_piece) *# remove the piece from the board*

self.\_\_board.\_white\_locations.pop(white\_piece) *# remove the piece from the list of black locations*

black\_options, black\_poison\_options = self.\_\_board.\_check\_options(self.\_\_board.\_black\_pieces, self.\_\_board.\_black\_locations, 'black')

white\_options, white\_posion\_options = self.\_\_board.\_check\_options(self.\_\_board.\_white\_pieces, self.\_\_board.\_white\_locations, 'white')

self.\_\_board.\_turn\_step = 0

self.\_\_board.\_selection = 100

valid\_moves = []

**if** event.type == pygame.KEYDOWN **and** game\_over: *# If the user wants to play again after the game is over, they can press enter and the game will restart.*

**if** event.key == pygame.K\_RETURN:

game\_over = False

self.\_\_board.\_initialise\_game\_variables()

valid\_moves = []

black\_options , black\_poison\_options = self.\_\_board.\_check\_options(self.\_\_board.\_black\_pieces, self.\_\_board.\_black\_locations, 'black')

white\_options , white\_poison\_options = self.\_\_board.\_check\_options(self.\_\_board.\_white\_pieces, self.\_\_board.\_white\_locations, 'white')

pygame.display.flip()

pygame.quit()

### **boardclass.py**

**import** pygame

**import** warnings

**from** pawnclass **import** Pawn

**from** bishopclass **import** Bishop

**from** serpentclass **import** Serpent

**from** kingclass **import** King

**from** queenclass **import** Queen

**from** empressclass **import** GrandEmpress

**class** Board(object):

**def** \_\_init\_\_(self):

self.\_turn\_step = 0

self.\_selection = 100

self.\_white\_promote = False

self.\_black\_promote = False

self.\_black\_promotions = ['bishop', 'serpent', 'empress', 'queen']

self.\_white\_promotions = ['bishop', 'serpent', 'empress', 'queen']

self.\_promo\_index = 100

*# check variables/ flashing counter*

self.\_counter = 0

self.\_white\_pieces = ['empress', 'serpent', 'bishop', 'king', 'queen', 'bishop', 'serpent', 'empress',

'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn']

self.\_black\_pieces = ['empress', 'serpent', 'bishop', 'king', 'queen', 'bishop', 'serpent', 'empress',

'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn']

self.\_piece\_list = ['pawn', 'queen', 'king', 'serpent', 'empress', 'bishop']

self.\_black\_queen = None

self.\_black\_queen\_small = None

self.\_black\_king = None

self.\_black\_king\_small = None

self.\_black\_empress = None

self.\_black\_empress\_small = None

self.\_black\_bishop = None

self.\_black\_bishop\_small = None

self.\_black\_serpent = None

self.\_black\_serpent\_small = None

self.\_black\_pawn = None

self.\_black\_pawn\_small = None

self.\_white\_queen = None

self.\_white\_queen\_small = None

self.\_white\_king = None

self.\_white\_king\_small = None

self.\_white\_empress = None

self.\_white\_empress\_small = None

self.\_white\_bishop = None

self.\_white\_bishop\_small = None

self.\_white\_serpent = None

self.\_white\_serpent\_small = None

self.\_white\_pawn = None

self.\_white\_pawn\_small = None

self.\_white\_locations = [(0, 0), (1, 0), (2, 0), (3, 0), (4, 0), (5, 0), (6, 0), (7, 0),

(0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1)]

self.\_black\_locations = [(0, 7), (1, 7), (2, 7), (3, 7), (4, 7), (5, 7), (6, 7), (7, 7),

(0, 6), (1, 6), (2, 6), (3, 6), (4, 6), (5, 6), (6, 6), (7, 6)]

self.\_font = pygame.font.Font('freesansbold.ttf', 40)

self.\_white\_material\_count = 0

self.\_black\_material\_count = 0

**def** \_initialise\_game\_variables(self):

self.\_white\_pieces = ['empress', 'serpent', 'bishop', 'king', 'queen', 'bishop', 'serpent', 'empress',

'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn']

self.\_white\_locations = [(0, 0), (1, 0), (2, 0), (3, 0), (4, 0), (5, 0), (6, 0), (7, 0),

(0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1), (8, 1), (9, 1)]

self.\_black\_pieces = ['empress', 'serpent', 'bishop', 'king', 'queen', 'bishop', 'serpent', 'empress',

'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn']

self.\_black\_locations = [(0, 7), (1, 7), (2, 7), (3, 7), (4, 7), (5, 7), (6, 7), (7, 7),

(0, 6), (1, 6), (2, 6), (3, 6), (4, 6), (5, 6), (6, 6), (7, 6), (8, 6), (9, 6)]

self.\_turn\_step = 0

self.\_selection = 100

**def** \_draw\_board(self, screen, hex\_colour1, hex\_colour2):

row\_count, col\_count = 8,8

cell\_size = 100

board\_colour1 = pygame.Color(hex\_colour1)

board\_colour2 = pygame.Color(hex\_colour2)

border\_colour = pygame.Color('gold')

**for** row **in** range(row\_count):

**for** col **in** range(col\_count):

x = col \* cell\_size

y = row \* cell\_size

*# Alternate between two board colours*

**if** (row + col) % 2 == 0:

pygame.draw.rect(screen, board\_colour1, pygame.Rect(x, y, cell\_size, cell\_size))

**else**:

pygame.draw.rect(screen, board\_colour2, pygame.Rect(x, y, cell\_size, cell\_size))

*# Draw the border around the board*

pygame.draw.rect(screen, border\_colour, pygame.Rect(0, 0, cell\_size \* col\_count, cell\_size \* row\_count), 5)

**def** \_work\_out\_offset(self, pieceset):

*# When drawing the pieces on the board, the images are coming in different sizes so when drawing them, we need to offset them so that they fit on each square.*

*# This offsetting process is done for every pieceset.*

**if** pieceset == "basic":

offset\_x, offset\_y = 5, 7

**elif** pieceset == "3D":

offset\_x, offset\_y = 0, -8

**elif** pieceset == "anarcandy":

offset\_x, offset\_y = 15, 8

**elif** pieceset == "merida":

offset\_x, offset\_y = 3, 3

**return** offset\_x, offset\_y

**def** \_draw\_pieces(self, screen, pieceset):

*#This function is responsible for drawing the pieces on the board.*

offset\_x, offset\_y = 0, 0

offset\_x, offset\_y = self.\_work\_out\_offset(pieceset)

**for** i, (pieces, locations, colour) **in** enumerate([(self.\_white\_pieces, self.\_white\_locations, 'red'), (self.\_black\_pieces, self.\_black\_locations, 'blue')]):

**for** j, piece **in** enumerate(pieces):

index = self.\_piece\_list.index(piece)

pawn\_offset\_x, pawn\_offset\_y = 10, 0 *# Additional offset for pawn*

**if** piece == 'pawn':

screen.blit(self.\_white\_pawn **if** colour == 'red' **else** self.\_black\_pawn, (locations[j][0] \* 100 + pawn\_offset\_x, locations[j][1] \* 100 + pawn\_offset\_y))

**else**:

screen.blit(self.\_white\_images[index] **if** colour == 'red' **else** self.\_black\_images[index], (locations[j][0] \* 100 + offset\_x, locations[j][1] \* 100 + offset\_y))

*# Draw selection rectangle*

**if** (self.\_turn\_step < 2 **and** i == 0) **or** (self.\_turn\_step >= 2 **and** i == 1):

**if** self.\_selection == j:

pygame.draw.rect(screen, colour, [locations[j][0] \* 100 + 1, locations[j][1] \* 100 + 1, 100, 100], 2)

**def** \_check\_options(self, pieces, locations, turn):

moves\_list = []

all\_moves\_list = []

poisoned\_list = [] *# Initialize poisoned\_list outside the loop*

**for** i **in** range(len(pieces)):

location = locations[i]

piece = pieces[i]

**if** piece == 'pawn':

pieceselect = Pawn()

moves\_list = pieceselect.\_check\_pawn(location, turn, self.\_white\_locations, self.\_black\_locations)

**elif** piece == 'empress':

pieceselect = GrandEmpress()

moves\_list = pieceselect.\_check\_empress(location, turn, self.\_white\_locations, self.\_black\_locations)

**elif** piece == 'serpent':

pieceselect = Serpent()

moves\_list, poisoned\_list = pieceselect.\_check\_serpent(location, turn, self.\_white\_locations, self.\_black\_locations)

**elif** piece == 'bishop':

pieceselect = Bishop()

moves\_list = pieceselect.\_check\_bishop(location, turn, self.\_white\_locations, self.\_black\_locations)

**elif** piece == 'queen':

pieceselect = Queen()

moves\_list = pieceselect.\_check\_queen(location, turn, self.\_white\_locations, self.\_black\_locations)

**elif** piece == 'king':

pieceselect = King()

moves\_list = pieceselect.\_check\_king(location, turn, self.\_white\_locations, self.\_black\_locations)

all\_moves\_list.append(moves\_list)

**return** all\_moves\_list, poisoned\_list

**def** \_check\_valid\_moves(self, white\_options, black\_options):

*# check for valid moves for just selected piece*

**if** self.\_turn\_step < 2:

options\_list = white\_options

**else**:

options\_list = black\_options

valid\_options = options\_list[self.\_selection]

**return** valid\_options

**def** \_draw\_valid(self, screen, moves):

*# draw valid moves on screen*

**if** self.\_turn\_step < 2:

colour = 'red'

**else**:

colour = 'blue'

**for** i **in** range(len(moves)):

pygame.draw.circle(screen, colour, (moves[i][0] \* 100 + 50, moves[i][1] \* 100 + 50), 5)

**def** \_check\_promotion(self):

*# This function is responsible for checking whether a white pawn has reached the eighth rank or a black pawn has reached the first rank.*

pawn\_indexes = []

white\_promotion = False

black\_promotion = False

promote\_index = 100

**for** i **in** range(len(self.\_white\_pieces)):

**if** self.\_white\_pieces[i] == 'pawn':

pawn\_indexes.append(i)

**for** i **in** range(len(pawn\_indexes)):

**if** self.\_white\_locations[pawn\_indexes[i]][1] == 7:

white\_promotion = True

promote\_index = pawn\_indexes[i]

pawn\_indexes = []

**for** i **in** range(len(self.\_black\_pieces)):

**if** self.\_black\_pieces[i] == 'pawn':

pawn\_indexes.append(i)

**for** i **in** range(len(pawn\_indexes)):

**if** self.\_black\_locations[pawn\_indexes[i]][1] == 0:

black\_promotion = True

promote\_index = pawn\_indexes[i]

**return** white\_promotion, black\_promotion, promote\_index

**def** \_draw\_promotion(self, screen):

*# When a pawn reaches the eighth or first rank, a menu will be drawn upon completion of the move, asking the user to select a piece to promote the pawn to.*

pygame.draw.rect(screen, 'dark gray', [800, 0, 200, 420])

**if** self.\_white\_promote:

colour = 'white'

**for** i **in** range(len(self.\_white\_promotions)):

piece = self.\_white\_promotions[i]

index = self.\_piece\_list.index(piece)

screen.blit(self.\_white\_images[index], (860, 5 + 100 \* i))

**elif** self.\_black\_promote:

colour = 'black'

**for** i **in** range(len(self.\_black\_promotions)):

piece = self.\_black\_promotions[i]

index = self.\_piece\_list.index(piece)

screen.blit(self.\_black\_images[index], (860, 5 + 100 \* i))

pygame.draw.rect(screen, colour, [800, 0, 200, 420], 8)

**def** \_check\_promo\_select(self):

*# Work out what the user wants to promote to.*

mouse\_pos = pygame.mouse.get\_pos()

left\_click = pygame.mouse.get\_pressed()[0]

x\_pos = mouse\_pos[0] // 100

y\_pos = mouse\_pos[1] // 100

*# Check if the promotion index is not None and the mouse position is within the promotion area*

**if** self.\_promo\_index **is** **not** None **and** y\_pos < 4 **and** left\_click **and** x\_pos >= 8:

*# Check if it's a white promotion and the indices are within bounds*

**if** self.\_white\_promote **and** 0 <= self.\_promo\_index < len(self.\_white\_pieces) **and** 0 <= y\_pos < len(self.\_white\_promotions):

self.\_white\_pieces[self.\_promo\_index] = self.\_white\_promotions[y\_pos]

*# Check if it's a black promotion and the indices are within bounds*

**elif** self.\_black\_promote **and** 0 <= self.\_promo\_index < len(self.\_black\_pieces) **and** 0 <= y\_pos < len(self.\_black\_promotions):

self.\_black\_pieces[self.\_promo\_index] = self.\_black\_promotions[y\_pos]

**def** \_setImages(self, black\_queen, black\_king, black\_empress, black\_bishop, black\_serpent, black\_pawn, white\_queen, white\_king, white\_empress, white\_bishop, white\_serpent, white\_pawn):

*# This function is responsible for initializing the images required as part of the graphical elements to my game.*

self.\_black\_serpent = pygame.transform.scale(black\_serpent, (95, 95))

self.\_black\_serpent\_small = pygame.transform.scale(black\_serpent, (45, 45))

self.\_black\_empress = pygame.transform.scale(black\_empress, (100, 100))

self.\_black\_empress\_small = pygame.transform.scale(black\_empress, (45, 45))

self.\_white\_serpent = pygame.transform.scale(white\_serpent, (100, 100))

self.\_white\_serpent\_small = pygame.transform.scale(white\_serpent, (45, 45))

self.\_white\_empress = pygame.transform.scale(white\_empress, (100, 100))

self.\_white\_empress\_small = pygame.transform.scale(white\_empress, (45, 45))

warnings.filterwarnings("ignore", category=RuntimeWarning) *# i kept getting a libpng error upon importing the images and saw that this was the best way to suppress it.*

self.\_black\_queen = pygame.transform.scale(black\_queen, (100, 100))

self.\_black\_queen\_small = pygame.transform.scale(black\_queen, (45, 45))

self.\_black\_king = pygame.transform.scale(black\_king, (100, 100))

self.\_black\_king\_small = pygame.transform.scale(black\_king, (45, 45))

self.\_black\_bishop = pygame.transform.scale(black\_bishop, (95, 95))

self.\_black\_bishop\_small = pygame.transform.scale(black\_bishop, (45, 45))

self.\_black\_pawn = pygame.transform.scale(black\_pawn, (100, 100))

self.\_black\_pawn\_small = pygame.transform.scale(black\_pawn, (45, 45))

warnings.filterwarnings("ignore", category=RuntimeWarning)

self.\_white\_queen = pygame.transform.scale(white\_queen, (100, 100))

self.\_white\_queen\_small = pygame.transform.scale(white\_queen, (45, 45))

self.\_white\_king = pygame.transform.scale(white\_king, (100, 100))

self.\_white\_king\_small = pygame.transform.scale(white\_king, (45, 45))

self.\_white\_bishop = pygame.transform.scale(white\_bishop, (100, 100))

self.\_white\_bishop\_small = pygame.transform.scale(white\_bishop, (45, 45))

self.\_white\_pawn = pygame.transform.scale(white\_pawn, (100, 100))

self.\_white\_pawn\_small = pygame.transform.scale(white\_pawn, (45, 45))

warnings.filterwarnings("ignore", category=RuntimeWarning)

self.\_white\_images = [self.\_white\_pawn, self.\_white\_queen, self.\_white\_king, self.\_white\_serpent, self.\_white\_empress, self.\_white\_bishop]

self.\_small\_white\_images = [self.\_white\_pawn\_small, self.\_white\_queen\_small, self.\_white\_king\_small, self.\_white\_serpent\_small, self.\_white\_empress\_small, self.\_white\_bishop\_small]

self.\_black\_images = [self.\_black\_pawn, self.\_black\_queen, self.\_black\_king, self.\_black\_serpent, self.\_black\_empress, self.\_black\_bishop]

self.\_small\_black\_images = [self.\_black\_pawn\_small, self.\_black\_queen\_small, self.\_black\_king\_small, self.\_black\_serpent\_small, self.\_black\_empress\_small, self.\_black\_bishop\_small]

### **piececlass.py**

**class** Piece(object): *# Base class for all of the pieces. The pieces all inherit the methods and attributes from this class*

**def** \_\_init\_\_(self):

self.\_value = None

**def** \_get\_friends\_and\_enemies(self, colour, white\_locations, black\_locations):

*# This function determines which are your own pieces based on the colour.*

*# I.e. if you have the white pieces and it's your turn, the enemy pieces are the black pieces (represented by black\_locations) and the friendly pieces are the white pieces (represented by white\_locations) and vice versa.*

**if** colour == 'white':

**return** black\_locations, white\_locations

**else**:

**return** white\_locations, black\_locations

### **serpentclass.py**

**from** piececlass **import** Piece

**class** Serpent(Piece): *# polymorphism shown here*

*# serpent piece class. This is the second new piece I created as part of my variant.*

**def** \_\_init\_\_(self):

super().\_\_init\_\_() *# inheritance shown here*

self.\_\_setAttributes()

**def** \_\_setAttributes(self):

self.\_value = 6

**def** \_\_getTargets(self, colour):

**if** colour == 'white':

targets = [(0, 3)]

**else**:

targets = [(0, -3)]

**return** targets

**def** \_check\_serpent(self, position, colour, white\_locations, black\_locations):

*# Get all of the legal moves that can be played by the serpent*

moves\_list = [] *# regular moves*

poisoned\_list = [] *# squares where enemy pieces are present.*

enemies\_list, friends\_list = self.\_get\_friends\_and\_enemies(colour, white\_locations, black\_locations)

targets = self.\_\_getTargets(colour)

**for** x, y **in** targets:

target\_x, target\_y = position[0] + x, position[1] + y

**if** 0 <= target\_x <= 7 **and** 0 <= target\_y <= 7 **and** (target\_x, target\_y) **not** **in** friends\_list:

moves\_list.append((target\_x, target\_y))

**if** (target\_x, target\_y) **in** enemies\_list:

poisoned\_list.append((target\_x, target\_y))

**return** moves\_list, poisoned\_list

### **empressclass.py**

**from** piececlass **import** Piece

**class** GrandEmpress(Piece): *# polymorphism shown here*

*# grand empress piece class*

*# This is the first new piece I have created as part of my variant.*

**def** \_\_init\_\_(self):

super().\_\_init\_\_() *# inheritance shown here*

self.\_\_setAttributes()

**def** \_\_setAttributes(self):

self.\_value = 7 *# In the game of chess, the king doesn't actually carry a value. Capturing the king or putting the king in checkmate signifies the game is over and by no means affects the material count.*

*# By giving the king an extremely large value (relative to the other pieces), we can work out whether the game is over by working out the overall material count for a given side.*

**def** \_check\_empress(self, position, colour, white\_locations, black\_locations):

moves\_list = []

enemies\_list, friends\_list = self.\_get\_friends\_and\_enemies(colour, white\_locations, black\_locations)

targets = [(2,1), (2,-1), (1,2), (1,-2), (-1,2), (-1,-2), (-2,1), (-2,-1)]

**for** i **in** range(len(targets)):

target = (position[0] + targets[i][0], position[1] + targets[i][1])

**if** target **not** **in** friends\_list **and** target **not** **in** enemies\_list **and** 0 <= target[0] <= 7 **and** 0 <= target[1] <= 7:

moves\_list.append(target)

**return** moves\_list

### **kingclass.py**

**from** piececlass **import** Piece

**class** King(Piece): *# polymorphism shown here*

*# the king piece class*

**def** \_\_init\_\_(self):

super().\_\_init\_\_() *# inheritance shown here*

self.\_\_setAttributes()

**def** \_\_setAttributes(self):

self.\_value = 1000 *# In the game of chess, the king doesn't actually carry a value. Capturing the king or putting the king in checkmate signifies the game is over and by no means affects the material count.*

*# By giving the king an extremely large value (relative to the other pieces), we can work out whether the game is over by working out the overall material count for a given side.*

*# check king valid moves*

**def** \_check\_king(self, position, colour, white\_locations, black\_locations):

moves\_list = [] *#Stores all of the legal king moves.*

enemies\_list, friends\_list = self.\_get\_friends\_and\_enemies(colour, white\_locations, black\_locations)

**for** x\_pos **in** [-1, 0, 1]: *# A king can move one square in any direction. This for loop focusses on the x position.*

**for** y\_pos **in** [-1, 0, 1]: *# This nested for loops looks at the y position*

**if** x\_pos == y\_pos == 0: *#If they are both equal to 0, the king makes no moves.*

**pass**

target = (position[0] + x\_pos, position[1] + y\_pos)

**if** 0 <= target[0] <= 7 **and** 0 <= target[1] <= 7 **and** target **not** **in** friends\_list: *#Here, we make sure that the final end position the king ends up on is on the board (i.e. on a square with coordinates between 1 and 7 for both the x and y coordinates).*

moves\_list.append(target)

**return** moves\_list

### **queenclass.py**

**from** piececlass **import** Piece

**from** bishopclass **import** Bishop

**from** rookclass **import** Rook

**class** Queen(Piece): *# polymorphism shown here*

*# The queen piece class*

**def** \_\_init\_\_(self):

super().\_\_init\_\_() *# inheritance shown here*

self.\_\_setAttributes()

**def** \_\_setAttributes(self):

self.\_value = 9

*# check queen valid moves*

**def** \_check\_queen(self, position, colour, white\_locations, black\_locations):

piece1 = Bishop()

moves\_list = piece1.\_check\_bishop(position, colour, white\_locations, black\_locations)

piece2 = Rook()

second\_list = piece2.\_check\_rook(position, colour, white\_locations, black\_locations)

**for** i **in** range(len(second\_list)):

moves\_list.append(second\_list[i])

**return** moves\_list

### **rookclass.py**

**from** piececlass **import** Piece

**class** Rook(Piece): *# Polymorphism shown here*

*# The rook piece class*

**def** \_\_init\_\_(self):

super().\_\_init\_\_() *# Inheritance shown here*

self.\_\_setAttributes()

**def** \_\_setAttributes(self):

self.\_value = 5

**def** \_check\_rook(self, position, colour, white\_locations, black\_locations):

*#This function is responsible for fetching all of the legal rook moves in a given position.*

*#The rook works in a very similar way to the bishop except that the rook moves horizontally or vertically.*

moves\_list = []

enemies\_list, friends\_list = self.\_get\_friends\_and\_enemies(colour, white\_locations, black\_locations)

directions = [(0, 1), (0, -1), (1, 0), (-1, 0)] *# down, up, right, left*

**for** x\_pos, y\_pos **in** directions:

start\_x, start\_y = position

chain = 1

continue\_checking = True

**while** continue\_checking **and** 0 <= start\_x + chain \* x\_pos <= 7 **and** 0 <= start\_y + chain \* y\_pos <= 7:

new\_position = (start\_x + chain \* x\_pos, start\_y + chain \* y\_pos)

**if** new\_position **not** **in** friends\_list:

moves\_list.append(new\_position)

**if** new\_position **in** enemies\_list:

continue\_checking = False

chain += 1

**else**:

continue\_checking = False

**return** moves\_list

### **bishopclass.py**

**from** piececlass **import** Piece

**class** Bishop(Piece): *# Polymorphism shown here*

*# The Bishop piece class*

**def** \_\_init\_\_(self):

super().\_\_init\_\_() *# Inheritance shown here*

self.\_\_setAttributes()

**def** \_\_setAttributes(self):

*# Set the piece value.*

self.\_value = 3

**def** \_check\_bishop(self, position, colour, white\_locations, black\_locations):

*#This function is responsible for fetching the legal moves that can be made by a bishop in a given position.*

*#The bishop is able to move along the diagonals through as many squares as desired until there is a blockade in the diagonal that is being pursued.*

moves\_list = []

directions = [(1, -1), (-1, -1), (1, 1), (-1, 1)] *#These are the basic direction vectors of the movement without any chain.*

enemies\_list, friends\_list = self.\_get\_friends\_and\_enemies(colour, white\_locations, black\_locations)

**for** x\_pos, y\_pos **in** directions:

current\_x, current\_y = position *#Keeps track of starting position.*

chain = 1

keep\_checking = True

**while** keep\_checking **and** 0 <= current\_x + chain \* x\_pos <= 7 **and** 0 <= current\_y + chain \* y\_pos <= 7: *#Ensures that the chain stays on the board.*

new\_position = (current\_x + chain \* x\_pos, current\_y + chain \* y\_pos) *#The new position from moving in a certain direction.*

**if** new\_position **not** **in** friends\_list: *#Ensuring that our own pieces aren't already on that speciifc square.*

moves\_list.append(new\_position)

**if** new\_position **in** enemies\_list:

keep\_checking = False *# We have reached a 'blockade' position so we simply append the move.*

chain += 1

**else**:

keep\_checking = False

**return** moves\_list

### **pawnclass.py**

**from** piececlass **import** Piece

**class** Pawn(Piece): *# Polymorphism shown here*

**def** \_\_init\_\_(self):

super().\_\_init\_\_() *# Inheritance shown here*

self.\_\_setAttributes()

**def** \_\_setAttributes(self):

*# This function sets the value of the pieces.*

self.\_value = 1

**def** \_check\_pawn(self, position, colour, white\_locations, black\_locations):

*# This function is responsible for fetching all of the legal pawn moves.*

moves\_list = []

**if** colour == 'white':

**if** (position[0], position[1] + 1) **not** **in** white\_locations **and** \

(position[0], position[1] + 1) **not** **in** black\_locations **and** position[1] < 7:

moves\_list.append((position[0], position[1] + 1))

*# indent the check for two spaces ahead, so it is only checked if one space ahead is also open*

**if** (position[0], position[1] + 2) **not** **in** white\_locations **and** \

(position[0], position[1] + 2) **not** **in** black\_locations **and** position[1] == 1:

moves\_list.append((position[0], position[1] + 2))

**if** (position[0] + 1, position[1] + 1) **in** black\_locations:

moves\_list.append((position[0] + 1, position[1] + 1))

**if** (position[0] - 1, position[1] + 1) **in** black\_locations:

moves\_list.append((position[0] - 1, position[1] + 1))

**else**:

**if** (position[0], position[1] - 1) **not** **in** white\_locations **and** \

(position[0], position[1] - 1) **not** **in** black\_locations **and** position[1] > 0:

moves\_list.append((position[0], position[1] - 1))

*# indent the check for two spaces ahead, so it is only checked if one space ahead is also open*

**if** (position[0], position[1] - 2) **not** **in** white\_locations **and** \

(position[0], position[1] - 2) **not** **in** black\_locations **and** position[1] == 6:

moves\_list.append((position[0], position[1] - 2))

**if** (position[0] + 1, position[1] - 1) **in** white\_locations:

moves\_list.append((position[0] + 1, position[1] - 1))

**if** (position[0] - 1, position[1] - 1) **in** white\_locations:

moves\_list.append((position[0] - 1, position[1] - 1))

**return** moves\_list

### **stack.py**

**class** StackException(Exception):

**def** \_\_init\_\_(self, value):

self.value = value

**def** \_\_str\_\_(self):

**return** self.value

**class** Stack(object):

**def** \_\_init\_\_(self, size):

self.\_\_TOS = 0

self.\_\_size = size

self.\_\_array = []

**def** isEmpty(self):

**return** self.\_\_TOS == 0

**def** isFull(self):

**return** self.\_\_TOS == self.\_\_size

**def** peek(self):

**if** self.isEmpty():

**raise** StackException("Stack is empty")

**else**:

**return** self.\_\_array[self.\_\_TOS - 1]

**def** push(self, item):

**if** self.isFull():

*# messagebox.askyesno(message="You seem to just be going in between screens! Taking you back to the homepage...")*

**raise** StackException("Stack is full")

**else**:

self.\_\_array.append(item)

self.\_\_TOS += 1

**def** pop(self):

**if** self.isEmpty():

**raise** StackException("Stack is empty")

**else**:

self.\_\_TOS -= 1

**return** self.\_\_array.pop()

**def** display(self):

**print**("Screen History Stack:")

**for** screen\_class **in** reversed(self.\_\_array[:self.\_\_TOS]):

**print**(screen\_class.\_\_name\_\_)

**def** clear(self):

self.\_\_array = []

self.\_\_TOS = 0

## **Data files**

### **blitzratings.txt**

04/02/2024

1709

05/02/2024

1709

07/02/2024

1716

08/02/2024

1716

### **bulletratings.txt**

04/02/2024

1901

05/02/2024

1901

07/02/2024

1901

08/02/2024

1901

### **rapidratings.txt**

04/02/2024

1801

05/02/2024

1801

07/02/2024

1801

08/02/2024

1801

### **classicalratings.txt**

04/02/2024

2216

05/02/2024

2216

07/02/2024

2216

08/02/2024

2216

## **General Implementation evaluation of my end user’s objectives from my own perspective**

I think that I have met the majority of my end user’s objectives in my implementation. For the ones that I haven’t been able to meet, some of them are down to design (such as the positioning of images and buttons and colour schemes). For instance, my end user initially wanted a neutral background and interface (see the questionnaire results I received in the Analysis section) but after consulting with them, they changed their mind about the user interface design so instead I decided in the eventuality that something like this were to happen with the square colours and other features, it would be difficult for my end user to change, and it wouldn’t suit their purpose. Therefore, I chose to make the background colours, piece sets and square colours customisable. The one other element I haven’t been able to achieve is the flip board function. After each move, my end user wanted me to create a functionality wherein the board would flip so that the user would be looking at the board from the perspective of the side whose turn it is. However, I had a lot of problems with this as I represented the board as a matrix which meant that the matrix locations themselves didn’t change but upon moving a piece, when it came to updating the position of the piece that has been moved, rather than moving the piece that has been moved, it would move the corresponding piece from the opponent’s side to the corresponding legal square and it would then be that same side’s turn. Therefore, I chose not to include this functionality in my implementation in my implementation. The other functionality I haven’t quite nailed is the serpent functionality. One of the serpents (the one on the right) for both the black and white pieces executes what its functionality was intended to perform, which is to poison the piece. If the empress is nearby (1 square away from the poisoned piece), the poisoned piece will not be removed from the board, and the empress and serpent are removed from the board. Otherwise, the poisoned piece is removed from the board. This functionality in my implementation only works in the serpent on the right and the serpent on the left seems to perform random moves albeit following the same vector movement that I have programmed into the system. Other than these objectives, I think that I have successfully achieved all of the other objectives and my testing videos below will demonstrate this.

# **Testing**

During the development process, it was of utmost importance that I ensured that the system worked as intended. Therefore, I chose to manage a testing system whereby after completing each section, I would then test the code to see whether it was successful and any other additional features that I might add to it.

For instance, when I was initially developing my user interface, I designed it with four buttons with one of them being the play against the Computer button, just to test that the system would look OK with the four buttons on them (though I knew that I wouldn’t actually need this for the actual end product itself because my end user doesn’t want the play against the AI component and would much rather I get the PGN notation for his game). I simply generated a blank screen to test my user interface should one of the buttons be pressed and as you can see from the screen recording below, this was extremely successful.

**https://drive.google.com/file/d/1xqJWPqrwBpVxGy-48EImoc3Auv-Awl4H/view?usp=share\_link**

(The letter after “48E” in the above link is a capital I and the letter after “-Aw” is a lowercase l)

This allowed me to work on areas that required improvements and had bugs so that I could deliver a better and more robust end product that is less likely to produce any errors.

## Post-development testing

With testing any software, there are three types of input testing:

* Normal testing – Acceptable data is inputted into the system and there should be some form of expected output.
* Erroneous testing – The data that is inputted into the system should cause some sort of error with the system as the data is not appropriate.
* Boundary testing – The data that is being inputted into the system is an edge case of the conditions and you test what will happen when the data only just meets the conditions or is just short of the condition (e.g. typos in the username field when a user is logging in).

Below are a series of links to YouTube videos to demonstrate the testing process I have taken and each video is for a specific aspect of the testing process (barring the first video, which combines the user interface and any features associated with that as well as the first part of the Standard Variant).

User Interface and Standard Variant (part 1) - **https://youtu.be/JHsbHLvmTC8**

Standard Variant (part 2) –**https://youtu.be/gNWz9QPEhpg**

Promotion functionality in the Standard variant: **http://tinyurl.com/3ypptcvu**

Big Bang Variant (and the length check functionality for the account creation system)- **https://youtu.be/DZKxWy335Ok**

In the testing video for the Big Bang Variant, I didn’t include the output from testing when the user wants to see the rules so I have attached the Google Drive link below to view this:

**https://drive.google.com/file/d/1xOdIPKc4L4vT-mZ\_AOSTCTVcvjcN7elm/view?usp=share\_link**

(In the above Google drive link, the letter following “1xOd” is not a small L, it is a capital I)

Promotion functionality in the Big Bang Variant - **http://tinyurl.com/24c9b6zb**

View Lichess Data Component (The video starts at 0:40 because of tech issues with the code editor I am using) - **https://youtu.be/mcFPhHZUilw**

# **Evaluation**

Below, for each requirement, I have listed the objectives of my project. The requirements highlighted green show that I’ve met the requirements of the end-user in that instance, the requirements that are yellow show that I’ve partially met these requirements and the requirements that are red show the requirements that I haven’t been able to meet.

## **Requirement 1**

1.     Create a user interface for the end-user to interact with the program.

1.1  The system must be accessible and easy to use.

1.2  The performance of the system must be similar to that of existing chess websites.

1.2.1       This objective will entail trying to include minimal lag into the program itself to mimic the existing chess websites.

1.2.2       Navigation through the program should be straight forward.

1.3  Any inputs and prompts that are required from the user must be self-explanatory.

1.3.1       There will be a sign-up button upon execution that will appear in the middle of the screen.

1.3.2       Two fields will appear here titled “Username” and “Password”

1.3.2.1   An account will be created provided the username doesn’t already exist in the database and the username and password are both greater than 6 characters.

1.3.3       There will also be a button at the bottom of this screen titled “Login” for those people who have already made an account.

1.3.3.1   They will be redirected to another page if this is pressed with two fields titled “Username” and “Password”.

1.3.3.2   The user inputs to these fields will be used to check the data stored in the text file to see whether there is any data matching the inputs.

1.4  The interface itself will be black and everything else will be in white to contrast against the dark backdrop.

1.4.1       My end-user wished for the appearance of the program to be “neutral”

1.4.2       The logo of my project will appear in the top left and top right of the screen.

1.4.3       The colour scheme will allow my end-user to be able to prepare for their tournaments regardless of time of day.

I think that I have successfully managed to meet the first requirement because I have met the majority of the requirements that I set out to achieve. Additionally, I found that the requirements that I wasn’t able to meet was down to my own design decisions in terms of how the user interface would look to the end-user. I didn’t want the screen to look too cluttered, so I decided against drawing the logos in the top left and top right-hand side of the screen and instead chose to have one in the middle of the screen and I didn’t draw the sign-up button in the middle of the screen as my end-user wasn’t a fan of it when I modelled.

There is very little lag between receiving some input from the user and an outcome from this input occurring, which is what my end-user wanted as he wants it similar to the performance of other chess websites like lichess.org and chess.com.

As I will explore below, I think that I have made navigation between the features extremely straightforward and made input prompts clear to the user to signal to them to input some data when required (as in the case of the login screen where I added a label saying ”Username:” and “Password” to indicate that these are the locations where the user need to input data.

Many of the requirements that are highlighted in yellow are simply down to the fact that it was a conscious design decision made at the time of implementation to make a smoother experience for my end-user. For instance, I didn’t follow the neutral colour scheme that my end-user was after simply because the user interface would look dull, boring and there would be no motivation for my end-user to keep playing and get better so that he can achieve his end-goal.

## **Requirement 2**

2.     Creating the board

2.1  After choosing one of the four options (Play against computer, 2-player chess, Big Ban g Chess, view previous games), a board will be created accordingly.

2.1.1       For the “Play against the Computer” option and the “2-player chess” option, the board will consist of 64 squares in an 8x8 configuration shaped like a square.

2.1.2       The variant I am created will use a board with a 10x10 configuration.

2.2  The user will be prompted to enter two colours from a list of colours that they would like for the squares on the board to be, which will be arranged in an alternating pattern.

2.2.1       On a standard chess board, there will be black and white colours with the queens in their own colour and the bottom right square of the board being located on the right side of the person playing with the white pieces.

2.3  The user will then be prompted to select a piece set from a list of piece sets which they will not be able to change during the game.

2.3.1       These pieces sets will come from both chess.com and lichess.org.

2.3.2       I will also try to make my own piece set, aimed specifically for the variant I am trying to create.

There is no ‘Play against the Computer’ option since my end-user decided to swap the AI program for a game PGN fetching program instead as he thought that it would suit his purposes far better. Barring that change, all of the other buttons exist. I didn’t use a 10 x 10 board configuration because the images that I am using to represent the pieces would have to be scaled to a much smaller size to stay on the square and you wouldn’t be able to differentiate between pieces, which is why I chose to stick with the 8x8 board for my variant. As for the rules about the piece positions, I have changed this rule slightly since I have added the customisability of the square colours and piece sets to my program so the 2.2.1 rules is no longer applicable, which is why I didn’t meet that requirement.

## **Requirement 3**

3.     Create chess pieces for both sides.

3.1  Regardless of the button pressed, chess pieces will need to be created.

3.1.1       This is where I will be relying on the dynamic generation of objects to create 8 pawns, 2 knights, 2 bishops, 2 rooks, 1 queen and 1 king for both the white pieces and the black pieces.

3.1.2       With the new variant I am creating, there will be one extra pawn, a serpent and the old woman on a 10x10 board configuration.

3.2  Pawn

3.2.1       Diagonal captures with en passant function, when two pawns are adjacent other and are laterally facing each other on the fifth or fourth rank (row), which is only available for the next move.

3.2.2       They can move one or two squares on the first move and then move one square at a time from then.

3.2.3       When the pawn reaches the other side, it can promote to a queen, bishop, rook or knight.

3.3  Knight

3.3.1       The knight moves in an L shape with one square in any direction (excluding diagonal movements) and then squares from the new square in a perpendicular direction to the initial movement.

3.3.2       They capture in the same way.

3.3.3       The only piece that is able to jump over its own army.

3.3.4       It changes the colour complex of the square it lands on every other move.

3.3.5       Knights can move backwards.

3.4  Bishop

3.4.1       Moves diagonally on the same colour complex.

3.4.2       It has no limits on the number of squares it can move provided the end square falls on the same diagonal as the initial starting square.

3.4.3       They capture in the same way they move.

3.4.4       Bishops can move backwards.

3.5  Rook

3.5.1       Rooks move up and down and left and right but can’t move diagonally.

3.5.2       There are no limits on the number of squares they can move provided the end square falls on the same file (column) or rank (row) as the initial square.

3.5.3       They move around the king when the king is being castled.

3.6  Queen

3.6.1       The most powerful piece that has a material value.

3.6.2       It can move in any direction but can’t jump over any pieces unlike the knight.

3.6.3       It combines the functionalities of the bishop and the rook.

3.7  King

3.7.1       The most powerful piece on a given chess board.

3.7.2       It is this piece that decides which side has won or lost in a given game of chess.

3.7.3       It can move one square in any direction and the only exception to that rule is in castling when the king moves two squares to the left or right.

3.7.3.1   Castling can only happen when the two squares the king moves to (either the left or the right) aren’t being controlled by an enemy piece (i.e., if there is a piece covering that square, the king would be in check so that makes the move illegal)

3.7.3.2   You can only castle when you haven’t moved the king or the rook that is concerned when castling.

3.8  Serpent

3.8.1       One of the new pieces I have introduced as part of my new variant.

3.8.2       It moves fours squares at a time and it can only move forward.

3.8.3       It has a special ability which fires a poison dart at an enemy piece and can only be used once in each game.

3.9  Grand Empress

3.9.1       Another piece that I have introduced as part of my new variant.

3.9.2       It moves exactly like the knight.

3.9.3       It has a special healing power where it removes the poison from a piece that has been attacked by the serpent provided it is neighbouring said piece.

3.10 I will prompt the user to enter a square colour and a piece set from a list of piece sets to choose from to design their board.

3.10.1 I will prompt the user to enter a square colour and a piece set from a list of piece sets to choose from to design their board.

3.10.2 These piece sets and square colours will come from lichess.org and chess.com.

3.10.3 I will be designing the piece set for the variant I am creating using graphics software.

I think that I have implemented this section well on the whole. However, there are some requirements that have been altered so that the pieces in question have a purpose throughout the whole game rather than just simply existing at the start. I would say that I’ve successfully implemented the pawns (the en passant and the promotion algorithms), the bishop, the knight, the rook, the serpent and the queen. I have nailed their functionalities exactly as they exist in the actual game of chess and my own variant. However, the problem lies with the functionalities of the king and the serpent. As I’ve demonstrated through my testing video for the Big Bang Variant, I had problems with changing the implementation of the serpents as I was only able to change the behaviour of one of the serpents. The other serpent wasn’t able to follow the same functionality, which was extremely problematic as that piece would malfunction upon moving to a square where there was already a piece present. It would display unprecedented behaviours such as capturing neighbouring pieces and I tried to resolve this issue but to no avail. With the king, I have successfully implemented the majority of its functionality but the one area I failed to change was the castling through check functionality and the castling while in check functionality. In the actual game, if your king is attacked by an enemy piece through a direct line of attack (i.e. the piece is ready to take the king on the next move if it isn’t moved), it is said to be in check and at this stage, you can’t castle according to the rules of the actual game. Castling is a move where the king moves two squares to the left or right and then the rook wraps around the other side and moves three squares to oppose the direction that the king moved initially. You also shouldn’t be able to castle through a check where if one of the two squares you are moving across also has the potential to have an enemy piece move to that square, you can’t make the castling manoeuvre. I had a lot of trouble trying to resolve this problem and, in the end, I was unsuccessful with this endeavour, but I hope it doesn’t detract from my end-user’s experience on my platform. Additionally, I wasn’t able to make an entire piece set but I made my own serpents and empresses uses graphics software so I have partially achieved this objective.

# 

## **Requirement 4**

4.     Making a move

4.1  The user will be prompted to make a move.

4.1.1       After the move is made, there will be a detection algorithm in place to see whether the move is legal and whether it is indeed possible (i.e., whether there is the specified piece on the board that can make it to that square).

4.1.2       If the move is legal, the piece will move accordingly, and the opponent makes their next move until checkmate is attained where the king has no legal squares to move to.

4.2  If the move is illegal, an error message will be returned to the user prompting them to play another move instead or the move will not be completed.

4.3  Once a move has been made, the board should flip to the perspective of the person’s turn.

4.3.1       This algorithm will reverse the matrix I will be using to represent the board and then displaying it.

Of all of the requirements I made for the project, I think that this was definitely the worst delivered. For the move making algorithms, I implemented things slightly differently. I didn’t design a prompt for the user to input their moves but rather just displayed the window and waited for them to input a move. After fetching the move, my requirements state that I must check whether the move is legal and if it is, the move should be played and if it isn’t, then an error message should be outputted. I implemented this section slightly differently. Firstly, I fetched all of the legal moves that can be played in a certain position before then asking the user to then input a move. If the move turned out to be legal, the move would get played but if the move isn’t legal, nothing would happen and the legal move highlighter would still point out the squares that a selected piece can move to.

The final sub-requirement of this requirement was the board flipping function. After finishing the majority of my implementation, I attempted to add the flip board function so that it would flip to the perspective of the side making the move. However, every time I tried this, I would encounter the same error. The code would run perfectly for the first couple of times before then malfunctioning. After flipping the board, since the matrix would be reversed, it meant that black’s pawns were able to move backwards on themselves. Additionally, in certain implementations of the flip board function, I found that instead of moving the piece selected to a given square, it would move the corresponding piece of your opponent’s piece to the corresponding square, which mimics that movement.

## **Requirement 5**

5.     Finding out whether there is a definite result.

5.1  One possibility is checkmate.

5.1.1       This is one of the key algorithms that I will have to incorporate within the program for all three of the features which my end-user has requested.

5.1.2       It will work by checking whether the kings are still on the board after a piece has been captured.

5.1.2.1   If the kings are on the board still, it hasn’t been captured so a checkmate hasn’t been established

5.1.2.2   If not, a checkmate has been established and the game ends.

5.2  Stalemate

5.2.1       For a stalemate to be achieved, one side must have no legal moves that can be played, and their king must not be in check.

5.2.2       This will again require the legal move finder algorithm which will find all the legal moves in a given position.

5.2.3       If there aren’t any legal moves stored within the list, a stalemate has been established and the game ends in a draw.

5.3 For the variant, there is no checkmate, en passant, check or castling. The game is simply to capture pieces with cooler functionalities.

**A screen shot of a computer program

Description automatically generated**I think that I have met the requirements here really well. There can only be three outcomes from a game: win, loss and draw.  Initially, I was thinking about adding a threefold repetition function here but I figured that since Vishnu would be playing by himself to analyse his own games and play against himself to see where his weaknesses lie, I wouldn’t really require such a function and without it, it would allow him to think things through. I also tried to visualise an approach I could take to instead develop a checkmate detecting algorithm rather than a king capture algorithm but I couldn’t find a way. Therefore, I chose to stick with the king capture algorithm.

## 

## **Requirement 6**

6.     Navigation between features

6.1  My end-user mentioned buttons in the questionnaire.

6.1.1       My program is going to centre on the use of buttons to ease navigation.

6.1.2       I am going to create four buttons after login with the four options.

6.1.3       The “View previous games feature” will read the data from the API and return the PGN notation of the games that have been played in the past on the lichess website.

6.1.4       There will also be buttons going to and fro between pages to ease navigation and there will be a quit button on the top right of the program to terminate the program.

I think that this is another area where I’ve done well albeit a couple of partial successes due to conscious design decisions made at the time of the Implementation phase. I think that I have successfully managed to create a navigation system through each of the windows using buttons and that it has met my end-user’s requirements in that sense. However, with the requirement that has been highlighted in yellow, I didn’t want to position the quit button in the top right corner because when I positioned things at the top, they looked really cluttered and it didn’t look great, which I’m sure my end-user would appreciate, is the only reason I didn’t meet this requirement fully.

## **Requirement 7**

7.     Drawing the rating data graphically

7.1  Fetching the rating data

7.1.1       This program will use the Lichess.org Application Programming Interface, which will allow me to use their services and algorithms to get the data I need.

7.1.2       I will have to ask for the API token of my end user because they are made private so I will have to detail how to get the API token and then send it to me.

7.1.3       I will have to run the methods associated with the account data like client.account.get()

7.1.4       Direct access to the application programming interface isn’t possible so I will have to use the Python client called berserk, which was developed by the lichess developers, to access the data.

7.1.5       The data that arrives is received in dictionary form, which is a data structure that stores a value and then another value associated with the first value, so I will have to store the received data in a JSON file before manipulating the data.

7.2  Storing and manipulating the data

7.2.1       Upon receiving the data, I will then manipulate the data that I receive so that I get the bits of data I need and then empty the JSON data file before the next request to see the rating data.

7.2.2       From organising the data, I will then use the dictionary methods to get the ratings I need (using key values to find the ratings).

7.2.3       After getting the exact ratings, I will then have to store the data for each time format in its own text file.

7.2.4       In addition to the rating, I will have to store the date that the rating was fetched so that I can

7.3  Drawing the graph

7.3.1       To draw the graph, I will then have to go through the text file that is storing the data.

7.3.2       I will plot the time that the rating was fetched on the x axis and then the rating on the y axis.

7.3.3       I will also have to have a different line for each variant so that there is no confusion in the graphical representation and then draw a key on the diagram so that it highlights to the user which variant each line represents.

This is another section where I have successfully met all of the requirements. I used the requirements here as a checklist to ensure that I’ve met all of them to make up for the fact that I didn’t meet some of my earlier requirements from other components of the project. Through spending time conducting my own research, I discovered and quickly got to grips with the Python client, berserk, for the lichess API. It allowed me to utilise the functions that are stored within the client to fetch the data. Furthermore, through fetching specific details about my end-user, I was able to tailor this more to his own requirements. I think that my end-user is extremely happy with this section, though I’m not 100% sure what use this will be to him.

# 

## **Requirement 8**

8.     Outputting the game PGN of my end user’s games

8.1  I will have to meet the requirements of 7.1 here as well but I will have to use a different function to get the PGN data.

8.1.1       Upon sending the request, I will receive all of the games played by a certain account and this can be from any player on the site, not just my end user, so I will have to parameterise the function to ensure that the data that is received is strictly relevant.

8.2  Parameterising the data

8.2.1       There are many parameters for the export games function, like start date, end date and player account, for exporting the games

8.2.2       I need to parameterise the function by my end-user’s account, the variant he’s playing and then the number of games (which my end-user has specified as 10 at a time).

8.3  Outputting the data

8.3.1       From the data I receive from the API request, I will then have to iterate through the list of games I receive and then output each game.

This is another section where I have met all of the requirements. This is one of the components that my end-user wanted me to create in replacement of the AI and I think I’ve done it well. Through the same thorough research as the research involved for requirement 7, I was able to pick up the necessary functions, methods and skills required to get the game data from my client’s own account before displaying the game data in PGN form.

## **My end-user’s thoughts**

A screenshot of a screenshot of a text

Description automatically generated

I really liked your end product of the project as it meets loads of my requirements. I don’t have to use these online sites that constantly keep having outages. So, moving onto feedback:

### The visual display

I’m really happy with the project on the whole. I didn’t really have any problems with the login system, except sometimes clicking on the things I enter my username and password into. You have to click exactly on the fields so that you can type into it but I’ve got used to it now so it’s not really a problem. I think that the positioning of the buttons themselves are slightly off as well but that isn’t really a major issue. It’s really easy to use and go through, which I wasn’t too sure about how it would turn out. I thought that the performance and how quickly it works would be really slow but it’s actually much faster than I thought, and I really like this about it. It’s really like playing on an actual chess website in that regard (though sometimes the buttons are a bit slow after clicking on them). I think that a lot of thought has been put into the colour scheme as well because from the questionnaire that you sent me, I know you were looking at the common times of day I play the game and the colour scheme of the app really allows me to play on this any time during the day and at night as well, which I really like about the app.

I really like the colour scheme chosen for the display and I really like the customisable features for the background, board colour and piece set for when the games start. It’s really a lot like the lichess.org and chess.com websites in that sense. The one thing that could improve with the display would be the images. The pictures themselves for the pieces aren’t of the best quality so I guess that’s one area that could improve. For instance, I can tell that you drew the grand empress piece, but the image itself isn’t of the best quality but I can just about make out what it is. I also really like that you haven’t increased the board size for the variant after adding new pieces in because when I’ve played different variants in the past on chess.com, I’ve found it really confusing as to what’s going on.

### Piece functionalities

As for the piece functionalities, you have done pretty well here. I would say that you have built the pawns (I was particularly impressed with you adding en passant and the promotion algorithms), knights, bishops, rooks and queens well. The king has been done well on the whole as well but there is the functionality problem of being able to castle through check, which I’m sure you tried to fix. Now, I don’t mind that this functionality isn’t there but sometimes, I do forget that I’m in check so I need to be more wary about it. As for the variant, I really like the new functionalities you have added for the serpent and the grand empress as it brings a new dimension to the game in terms of healing powers, but the movement of the serpent is quite restricted so I think that the game would be even better if the serpent movement wasn’t as restricted. Also, I notice that one serpent piece moves like you’ve described it but the other one doesn’t. I know I let you create the variant without any input from myself but it would have been nice to have been able to have more new pieces. At first, I didn’t understand the purpose of the variant itself because I couldn’t identify any underlying goal (since I was able to capture the king) but I guess you designed it like for entertainment purposes.

### Navigation

When you asked me to fill in the questionnaire, I remember one of the questions was to do with what sort of appearance did I want my display to have and I really wanted buttons instead of the same old drop down menus you see on literally every chess site that is currently existing. I personally find them too sensitive. I think you’ve done really well with the buttons and the movement between displays is a lot faster than I actually thought it would be and I think you’ve done really well here.

### Making a move

I am relatively happy with the way that the moves are being made. I think that there are a couple of problems here, which are that I’m only able to see the board from the perspective of the black pieces and that there is no board flipping between moves. I think that this would have really separated your app from the currently existing chess websites as they don’t have a continuously flipping board after each move is made. I really like the piece highlights and the colours used to make it clear what the legal moves are for each piece. Some sound wouldn’t go amiss here after each move is made, but I think that’s more down to the fact that I’m used to playing online chess. That being said, I think it’s a really good app for preparation for over the board chess, because there are no sounds, and there is also no functionality to draw arrows to highlight potential plans that can be made during the game.

### Ending the game

This is another section I think you’ve done really well. As soon as the king is captured, the game ends and I really like it. However, one thing that could be improved about this would be checking whether a checkmate has been played because in actual chess games, both online and over the board, the games go on until checkmate rather than actually capturing the king. This is just a small thing and I don’t actually think it’s that big of a problem and it doesn’t actually detract anything away from the rest of the app.

### Drawing the ratings graph and fetching the PGN notation of my games

After getting rid of the AI program idea because I knew that I would be able to find better AIs to play against elsewhere to improve, I decided that I would much rather have a way of getting my online games in PGN format so that I can review them. Now, upon reviewing your program, I would have preferred for all of the components to be part of one program so I don’t have to go to a different file to access this functionality. However, concerning the component itself, I’m really happy with it. I’m able to look through my games and see where I went wrong and you’ve even managed to get the games by the time format played, which is even better because the only games I actually study are the rapid time format games I play. By far the bit I was most impressed with was the graph. I asked for an extra component of your choice after scrapping the AI idea and the graph idea is really cool! It gives me a good idea of how I’m playing currently and how riskily I’m currently playing (as I’ve recently discovered that as the time format decreases, I’m more likely to take greater risk with my moves). I also really like the fact that it’s dynamically generated as well. As soon as I press the button, it will fetch my ratings and then graph it out. This is definitely the most useful component of the three (the normal chess game, the variant you created and this) and I really like it.

# 

## **My comments about the end-user’s feedback**

### The visual display

With regards to clicking the buttons themselves, I don’t really have a choice as to where the detection algorithm for detecting whether a button has been clicked should check for mouse clicks, since I simply utilised the built-in Button class in the tkinter library to meet my own purposes. The buttons being a bit slow after clicking on them is yet again something I have no control over since the interaction of these buttons is controlled by the tkinter user interface library. I’m really happy that I was able to meet my end-user’s requirements in this instance. I know that you initially were set on having a neutral colour scheme but in the eventuality you ended up going against it, I had this implemented as a backup mechanism. I completely understand where my end user is coming from with regards to the picture quality. After creating the pictures, I tried running the images through an AI upscaling tool for better clarity but it still didn’t do much. If anything, it made the images worse, which is the reason why some of the images, particularly in the variant game, are slightly pixelated.

### Piece functionalities

I’m happy that I could meet my end-user’s requirements in terms of the piece functionalities on the whole. With the castling through check issue, I’ve tried implementing solutions to the problem but to no avail. I tried to design a smoother system as possible so that my end user would have a better experience and wouldn’t feel as though he was compromising in his preparation whilst simultaneously solving the other problems he had with the currently existing systems. As I already mentioned in my testing video, there was a problem with the second serpent on the left hand side because it wasn’t following the intended functionality which the one on the right hand side was following. I tried to fix this error but I couldn’t identify the logic error that was causing this problem. I also thought about the addition of newer and more complex piece functionalities, but I couldn’t think of any new ones I could add to the game so in the end, I decided against it.

### Navigation

I’m happy that my end-user is happy with the navigation tool. I agree with my end user that occasionally, the buttons do tend to lag a bit as they are executing their functionalities (particularly those on the Main Menu Screen directing the user to the options, which you can see in my testing videos). However, as I’ve already mentioned, I have no control over the functionality of the buttons because it is already created for me and I am simply utilising this object from the library I used to develop my user interface. Making my own button class to deal with user interactions on the screen would require in-depth study of the tkinter module, which I used to design my user interface, and that would require a lot more time so I would say that if I had a lot more time, I would improve upon the button functionality in the code so that the operations are executed much faster.

### Making a move

**A screen shot of a computer program

Description automatically generated**I’m happy that making the move is working for my end user. I think that I’ve met all of the requirements for this barring the flip board function. My end user didn’t inform me about the arrow functionality on the other chess websites and upon looking at the current systems, I didn’t come across this feature, but my end user didn’t require it anyway and based off their feedback, they are happy without it. I tried to develop a flip board function (shown below) but it simply didn’t work out for me, so I stuck with my original implementation of the code without the flip board function.

### Ending the game

I take on board my end-user’s feedback and agree with them that I could have changed the game ending algorithm to check for checkmate, but during the Design process, I found it far easier to implement it to check the presence of the king rather than checking for checkmate as I would have to do this in addition to having lots of other components involved. However, when it came to thinking about how I was going to implement the checkmate algorithm, I wasn’t really sure as to how I was going to achieve this. I got this idea of checking whether the king has been captured from the codewithfaraz.com tutorial (which I have linked in my bibliography below) and so I chose to stick with this for simplicity’s sake and trying to come up with a better solution to the problem.

### Drawing the ratings graph and fetching the PGN notation of my games

Initially, I was unsure about how to approach this but through thorough research being conducted, I quickly got to grips with this. I think that modelling this situation (as shown in the analysis section) really allowed me to get to grips with this API fetch and through my end user’s feedback, I can see that he is happy with this component and like him, I too think that I have successfully met this requirement and component of the program. I’m happy to see that the program has worked, and it is of some use to him. The one other thing I would have liked to have done in this case would have been to integrate this into the main program itself but as I’ve already mentioned in the documentation, there would be a huge class between the graphics libraries I am using (tkinter for the user display, PyGame for the game components and matplotlib for graphing the ratings) which would mean that I wouldn’t be able to execute the functionality that my end user is after. Therefore, I had to split the code into separate files so that I could achieve both functionalities, though I agree it would have been much better if it’s possible for me to integrate the View Lichess Stats functionality into the main program itself. What also helped me with this component of it was the familiarity with handling text files and familiarity with the matplotlib through some independent projects of my own.

# **Bibliography**

|  |  |
| --- | --- |
| References | Inspiration with regards to the project |
| lichess.org. (n.d.). *Source Code • lichess.org*. [online] Available at: https://lichess.org/source [Accessed 25 May 2023]. | I started to look into the proof of concept for my idea and whether it would indeed be possible to build the program. I found out that the major chess website, lichess.org, is an open-source software and found out that it is in fact written in the object oriented and functional programming language, Scala. |
| Wikipedia. (2022). *Chess.com*. [online] Available at: https://en.wikipedia.org/wiki/Chess.com.  ‌ | Chess.com is the other major chess website which has been written in a mixture of PHP, JavaScript and Java. It uses PHP for the main site, Java on the back end of the live server and JavaScript and HTML5 for the user interface. |
| ‌ Maranan, M. (n.d.). *How to Make a Chess Game with Pygame in Python - Python Code*. [Online] www.thepythoncode.com.  ‌ | Also, part of my proof of concept, when I discovered that it would be possible to program the game in Scala, I figured that it is highly probable that it would be possible to program in Python since it too supports object orientation. I sought a program that already existed to prove this, which then led me to take on the project. |
| The Big Bang Theory Wiki. (n.d.). *Three-Person Chess*. [online] Available at: https://bigbangtheory.fandom.com/wiki/Three-Person\_Chess. | I decided to make the game familiar to my end-user so I decided to slightly alter the game created by Sheldon so that it can be played on a standard chessboard, albeit a 10x10 board. |
| www.youtube.com. (n.d.). *Pygame in 90 Minutes - For Beginners*. [online] Available at: https://www.youtube.com/watch?v=jO6qQDNa2UY.  ‌ | I used this video created by TechWithTim to familiarise myself with the PyGame module as I figured that it would come in handy when it came to the implementation phase of the project. |
| www.youtube.com. (n.d.). *How To Play Chess: The Ultimate Beginner Guide*. [online] | This is the main video I used to get to grips with rules of chess so as to ensure that my end-user gets a smooth an experience as I can possibly provide. |
| database.lichess.org. (n.d.). *Lichess.org open database*. [online] Available at: https://database.lichess.org/.  ‌ | This is the database of rated games which I will be using as a backup for fetching the necessary games for the artificial intelligence program to reflect the ability of a club level player. I will only use this should I not be able to access the required rated games from the Lichess API. |
| www.cs.kent.ac.uk. (n.d.). *pgn-extract: Portable Game Notation (PGN) Manipulator for Chess Games*. [online] Available at: https://www.cs.kent.ac.uk/people/staff/djb/pgn-extract/.  ‌ | With the rated games that I have downloaded, I figured that I would receive several thousands of games without any separation, so I needed a way to parse the data. I tried to find some software online that would parse the pgn that I would receive for me. There were some results that came up and this will allow me to split the data into individual games so that the AI will be able to read the games and then mimic the playing style. If this doesn’t work, I plan on going through each game manually and separating each game with a unique character that is, statistically speaking, highly unlikely to appear in the PGN notation of the games before using the .split() method in python. |
| Team (CHESScom), C. com (2022). *Chess.com Officially Acquires Play Magnus, Carlsen Signs As Ambassador*. [online] Chess.com. Available at: https://www.chess.com/news/view/chesscom-acquires-pmg [Accessed 29 May 2023].  ‌ | This link was used to acquire information about chess.com acquiring PlayMagnus. |
| chess24.com. (2011). *chess24.com your playground*. [online] Available at: https://chess24.com/en.  ‌ | This is one of the systems I explore when I look at existing systems which my end-user could potentially make good use of when preparing for tournaments |
| Wikipedia. (2023). *Free Internet Chess Server*. [online] Available at: https://en.wikipedia.org/wiki/Free\_Internet\_Chess\_Server [Accessed 29 May 2023].  ‌ | This is one of the links I used to access information about one of the systems I explore. |
| www.freechess.org. (n.d.). *Free Internet Chess Server (FICS)*. [online] Available at: https://www.freechess.org/ [Accessed 29 May 2023].  ‌ | This is the other link I accessed when looking at FICS |
| freeCodeCamp.org (2017). A step-by-step guide to building a simple chess AI. [online] freeCodeCamp.org. Available at: https://www.freecodecamp.org/news/simple-chess-ai-step-by-step-1d55a9266977/.  ‌ | This is a webpage that I found on the website I have previously completed courses on that explains the basic steps required to make a chess AI. |
| berserk.readthedocs.io. (n.d.). *Welcome to berserk’s documentation! — berserk 0.10.0 documentation*. [online] Available at: https://berserk.readthedocs.io/en/master/. | In order to enable one of the features of my application, I used the Python client to access the Lichess.org API |
| Code with Faraz - Best Front-End Components and Blogs. (n.d.). *Create a Chess Game in Python Step-by-Step (Source Code)*. [online] Available at: https://www.codewithfaraz.com/python/19/create-a-chess-game-in-python-step-by-step-source-code [Accessed 9 Feb. 2024].  ‌ | This is the tutorial I followed to develop the standard chess variant. |