DE LA SALLE UNIVERSITY - MANILA

CHESS CONSOLE: A SIMPLE CHESS GAME WITH PYTHON ARRAYS

A Term Project

Presented to Mr.Ramon Stephen L. Ruiz

In Partial Fulfillment of the

Requirements for the Course Programming Logic and Design (PROLOGI)

---

by

TY, Josh Angelo - Signature/Initials

TAN, Irabelle Pristine - Signature/Initials

BLAS, Harold Sebastian - Signature/Initials

EQ3

April 20, 2023

TABLE OF CONTENTS

PROLOGI

I. Introduction

Chess is one of the oldest and most popular board games in the world, known for its strategic depth and complexity. It has been played for centuries, and its rules have remained largely unchanged. With the advent of modern technology and programming languages, chess can now be recreated in a digital form, allowing players to enjoy the game online, against artificial intelligence opponents, or even against other players remotely, without the need for a physical chess set.

In recent years, Python has emerged as a popular programming language for game development due to its versatility and ease of use. Python provides a wide range of libraries and tools that make it suitable for creating complex applications, such as a chess game. This project aims to leverage the power of Python to develop a fully functional chess game that provides an engaging and interactive experience for players.

A. Background of the Study

Chess has a rich history that spans over a millennium, with its origins tracing back to ancient India and its evolution from Chatarung into the modern game we know today. Throughout the centuries, chess has been studied, analyzed, and played by millions of people around the world, making it one of the most thoroughly researched and documented games in existence.

With the rise of computer technology, chess has also become a fertile ground for research and development in artificial intelligence (AI) and computer programming. In 1997, IBM's Deep Blue famously defeated the reigning world chess champion, Garry Kasparov, marking a significant milestone in the field of AI and showcasing the potential of computers to play chess at a high level. Since then, computer chess programs have continued to evolve and improve, with

advanced algorithms, powerful hardware, and sophisticated heuristics enabling AI opponents to challenge even the most skilled human players. Python, as a versatile and widely-used programming language, has also gained popularity in the field of computer chess programming due to its extensive libraries, ease of use, and community support.

The background of this study is based on the motivation to develop a chess game using Python as a final project, leveraging the rich history and complexity of chess, the advancements in AI and computer programming, and the versatility of Python as a powerful tool for game development. This research project aims to contribute to the growing body of knowledge in computer chess programming and provide insights into the implementation of a fully functional chess game using Python, incorporating elements such as correct move notation, graphical rendering, and optimization techniques. The results of this research can be used as a valuable resource for students, researchers, and game developers interested in the application of Python in creating chess games and other interactive applications.

B. Problem Statement

This project aims to develop a chess game with the use of Python that encompasses the complexities of a chess game, including game state representation, move generation, and user interface design. The key challenges of this project include the following:

1. Game state representation: Designing an efficient and accurate representation of the chessboard and the game state, including the position of all chess pieces, their legal moves, and the current game status.
2. Move generation: Developing an algorithm for generating valid moves for each type of chess piece based on the rules of chess, including handling special moves such as castling, en passant, and promotion.
3. User interface design: Creating a visually appealing and user-friendly graphical user interface (GUI) for the chess game, including features such as game setup, move input, game progress tracking, and game termination.

C. Objectives

C.1 General Objective

1. Develop a fully functional and visually appealing chess game using Python that accurately represents the complexities of chess gameplay, including game state representation, move generation, and user interface design.
2. Implement a user-friendly graphical user interface (GUI) that allows players to interact with the chess game intuitively, including features such as game setup, move input, game progress tracking, and game termination, providing an enjoyable gaming experience.
3. Implement user profiles and leaderboards that allow players to see their scores.

C.2 Specific Objectives

1. Design and implement an efficient and accurate game state representation in Python that captures the positions and movements of all chess pieces on the chessboard, along with the current game status, such as player turn, castling rights, and en passant possibility.
2. Develop a move generation algorithm that generates valid moves for each type of chess piece based on the rules of chess, including handling special moves such as castling, en passant, and promotion, while ensuring legal move generation and avoiding move duplication or omission.
3. Allow user login or signup, which follows with the choice to start the game or see the leaderboard first.
4. Develop a leaderboard which contains the user profiles and their corresponding points gained through winning matches.

D. Significance of the Project

This project has significant educational value as it provides a valuable learning experience for the programmers to enhance their programming skills in Python, including game development, user interface design, and testing/validation. Additionally, chess is a popular and

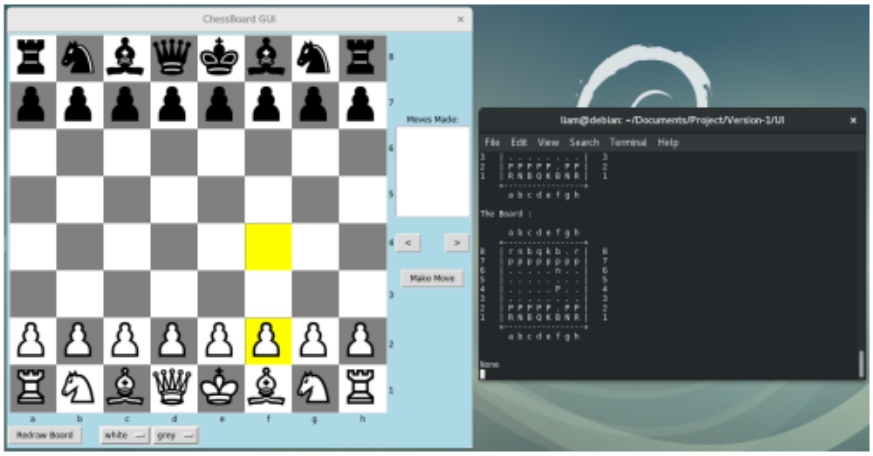
mind-stimulating game. Developing this in Python provides a practical application of programming concepts in a real-world context, allowing the programmer to apply the lessons they have learned to create a functional game that can be enjoyed by others.

II. Review of Related Literature

Python Chess Program Using Arrays

Chess can be made in various ways in Python, in fact it is quite easy to do if one utilizes specific modules made just for chess such as python-chess. These modules, such as python-chess, allow users to import the module to play or even create chess games by programming move generation and validation into the module, as well as support for different chess variants (Fiekas, 2022). There are also known chess sites/applications that are programmed using Python, a famous example being PyChess which is known for its huge selection of chess variants ranging from ASEAN Chess to Atomic Chess, and even supporting other chess-like games like Shogi or Makruk (PyChess, n.d.).

Now that it is known that chess can be made in Python, is it possible to program chess in Python by basing it on arrays? In Vallance's (2018) paper on *Exploring the Python Chess Module*, it can be seen that in the 4th figure of their paper in the 3rd part titled "Progress", that they are utilizing a board mechanism similar to an array.



Vallance (2018) created the orientation of their board by scanning a string separated by a slash symbol "/" to store the values of each row on the chess board. Due to that nature, it is possible to create this using an array by assigning each character in an array value instead of a string. After all, according to W3Schools (n.d.), Python strings are just an array of bytes that each represent a certain unicode character. Since their natures are so similar in python, scanning an array can be akin to scanning a string for each of the character values, making it possible to create a Python program for chess that mainly uses arrays.

In the paper titled *Autonomous Chess Playing Robot* by Rath et al. (2019), they utilize a numerical array for creating and generating a binary position map for chess pieces, with 1s for occupied spaces and 0s for unoccupied spaces. By combining the nature of Python strings acting like arrays, and the ability to map chess locations using a numerical array, it is now theoretically possible to create an array that stores the unicode of each chess piece in a location map. Due to this theoretical possibility, the only thing that one needs to make it practical is a Python module that has the nature of a numerical array but can store characters in each array location instead of integer values. In the program titled *Simple Python Chess*, Mitev (2011) utilizes chess notations to move around the chess pieces by first checking the initial location of the input (the first 2 characters in a move input, represents the chess piece that will be moved) and then checking the final location of the input (the last 2 characters in a move input, represents the location that the initial chess piece will attack or be moved to). It is possible to utilize this chess notation type of move input to make moving chess pieces around the board more easier, as it is possible to create a copy of the array and utilize the nature of parallel arrays to tie the values of the chess piece and their location on the board (Ruiz & Pakzad, 2023).

CharArrays, a type of array found in the numpy module of Python that notably allows for a way to interact and manipulate arrays of string and/or unicode type data in an intuitive manner (NumPy, n.d.). This means that by utilizing the characteristics of CharArray to store string/unicode values in a numerical array, the nature of numerical arrays to be able to be used as location maps, and the nature of strings/unicode values to store chess piece data, it is possible to

combine these truths to create a fully functioning chess game in Python by mainly utilizing arrays.

CharArray was also specifically picked due to the ability to store unicode values, which is more extensive than the standard alphanumeric values of a keyboard which is usually what a user keyboard input is composed of. CharArrays can output unicode data by changing the default value of the argument unicode=False to unicode=True so that it recognizes unicode strings and fetches their unicode characters from the computer's storage by their values (NumPy, n.d.). This allows for the display of chess piece images without having the need to know Graphic User Interface (GUI) due to the nature of unicode characters. Instead of scanning the string value of an array location and assigning an img to output on the GUI corresponding to the array value, it is possible to output a unicode character that shows a chess piece image instead by using the nature of CharArrays to accept unicode values.

Chess Features

While chess is mainly known for the board and the game itself, a chess game application still comprises different features such as keeping time for each player, an account system, and whatnot. Here it is possible to improve the game even more by adding quality of life features like this that add more to the game.

The first feature that can be added to an array-based Python chess program is time by using the time module of Python. The time.sleep() function can be utilized to simulate a countdown clock by using its nature of stopping execution of a certain task by the amount of seconds in the argument (Python, 2023c). The thread module and queue module of Python can be used in conjunction with this to only countdown while waiting for user input, which is similar to real life while a player thinks about their moves while the chess clock counts down and waits for the current player to press it. The thread module can be used to run a clock function in parallel to a user input function due to its multiprocessing nature (Python, 2023b). Meanwhile, the queue

module can be used to fetch the data from these threads and send them to the main function/program, allowing the main program to use these values to update variables such as getting and assigning the move input and remaining time to variables in main (Python, 2023a). Another module that can be utilized is the getpass module of Python, which is used for the input of security credentials like the password of an account system, this can help strengthen security while inputting account credentials (Satyam, 2020).

It is through these different modules and previous research that it can be safely said that it is possible to create a Python program that creates a chess game utilizing arrays, with the included features one would see in a standard chess game in real life like countdown timers and password protection.

III. Methodology

The following are the frameworks and/or charts that were used for the creation of the code. This includes an initial IPO (Input-Process-Output) chart to analyze the inputs of the user, how the data will be processed, and what will be outputted from these processes. The hierarchy chart connects all the functions and processes so it can be known where and how functions can be properly called upon in the code. The flowchart shows a more detailed version of the hierarchy chart to show the sequential version of the code. Meanwhile, the pseudocode follows the flowchart but is more easily readable by using the English language as a syntax.

A. Conceptual Framework – IPO Chart (Input-Process-Output-Chart)

Input -

Player 1 username & password (login or signup);

Player 2 username & password (login or signup);

Player 1 move using chess notation (repeat input while game has not ended or if input is invalid);

Player 2 move using chess notation (repeat input while game has not ended or if input is invalid);
Draw or Resign offers

Process -
If Player is logging in and information aligns with an account in the account database, then show stats;
If Player is logging in and information does not align with an account in the account database, then ask for input again;
If Player is signing up and username does not align with an account in the account database, then show stats;
If Player is signing up and username aligns with an account in the account database, then ask for input again;
While game has not ended(A Player's time has not reached 0, both kings on the board):
{
Countdown time (initial time of 10 minutes, counts down per second while there is no input, 5 second grace period before start of countdown);
Ask Player for move input using chess notation (may also draw or resign);
If draw: Ask both Players if they want to draw, if yes then end game and add draw to their stats, if no then continue game;
If resign: add loss to resigning Player's stats and add win to other Player's stats;
If move input is not valid: Ask input until move input is valid;
If move input is valid: do chess move;
Update chessboard with Player move inputs;
Update time;
Check if any game end scenarios have been reached, if yes then end game, if no then continue;
}(Cycled between the 2 Players)

Output -
Player stats;
List of possible moves;
Current time per player (updates and prints every second (replacing));

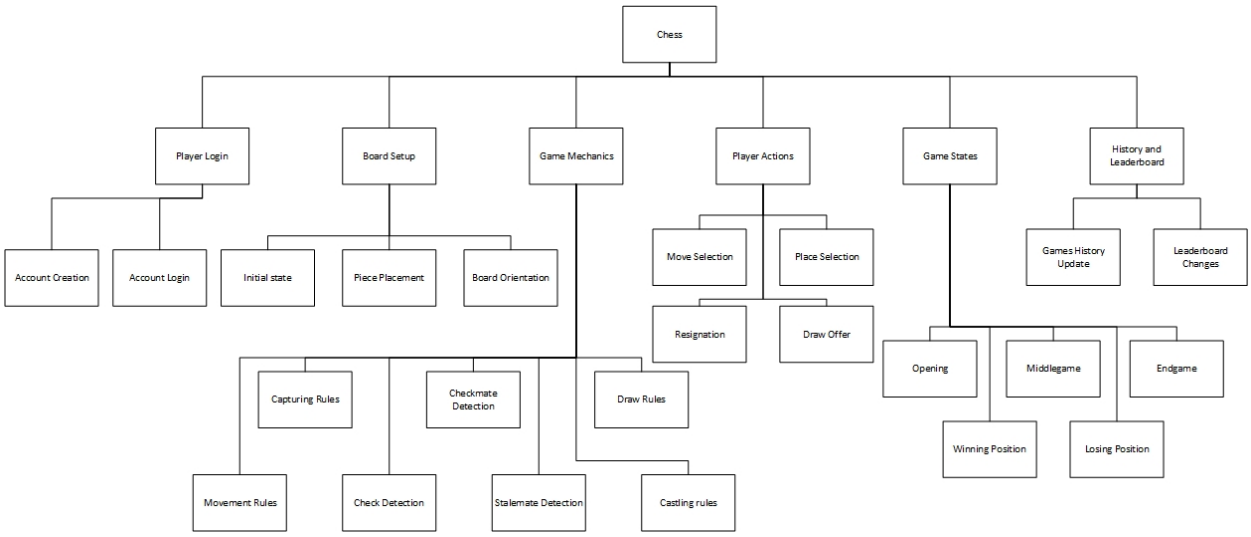
Current chessboard (updates and prints after every move);

Winner of chess game;

Leaderboard according to account database wins;

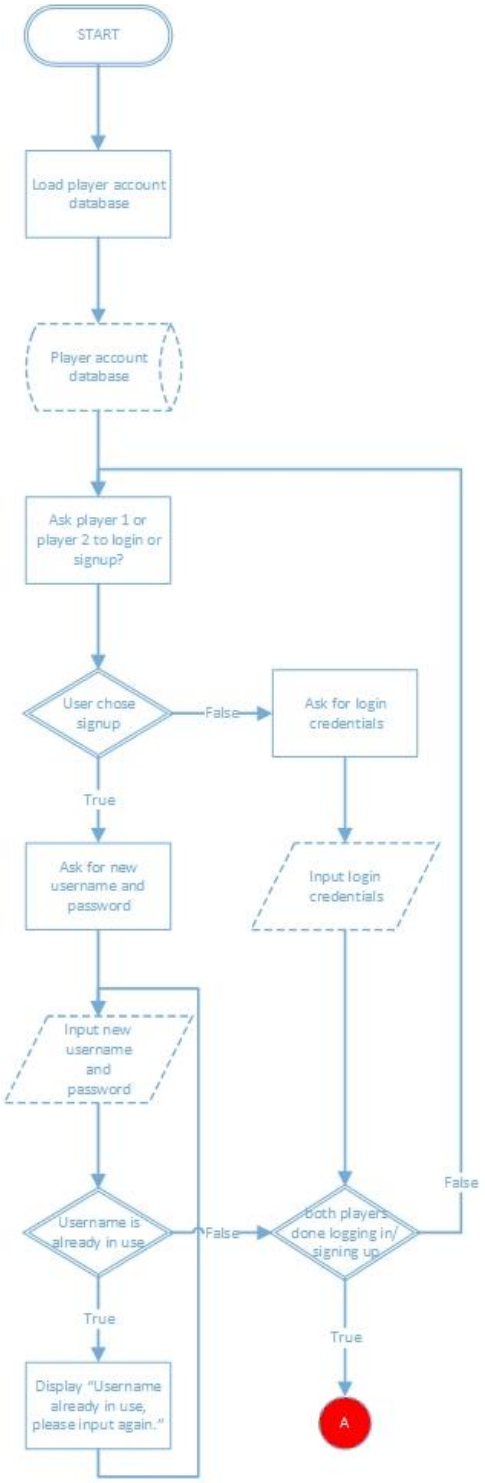
B. Hierarchy Chart

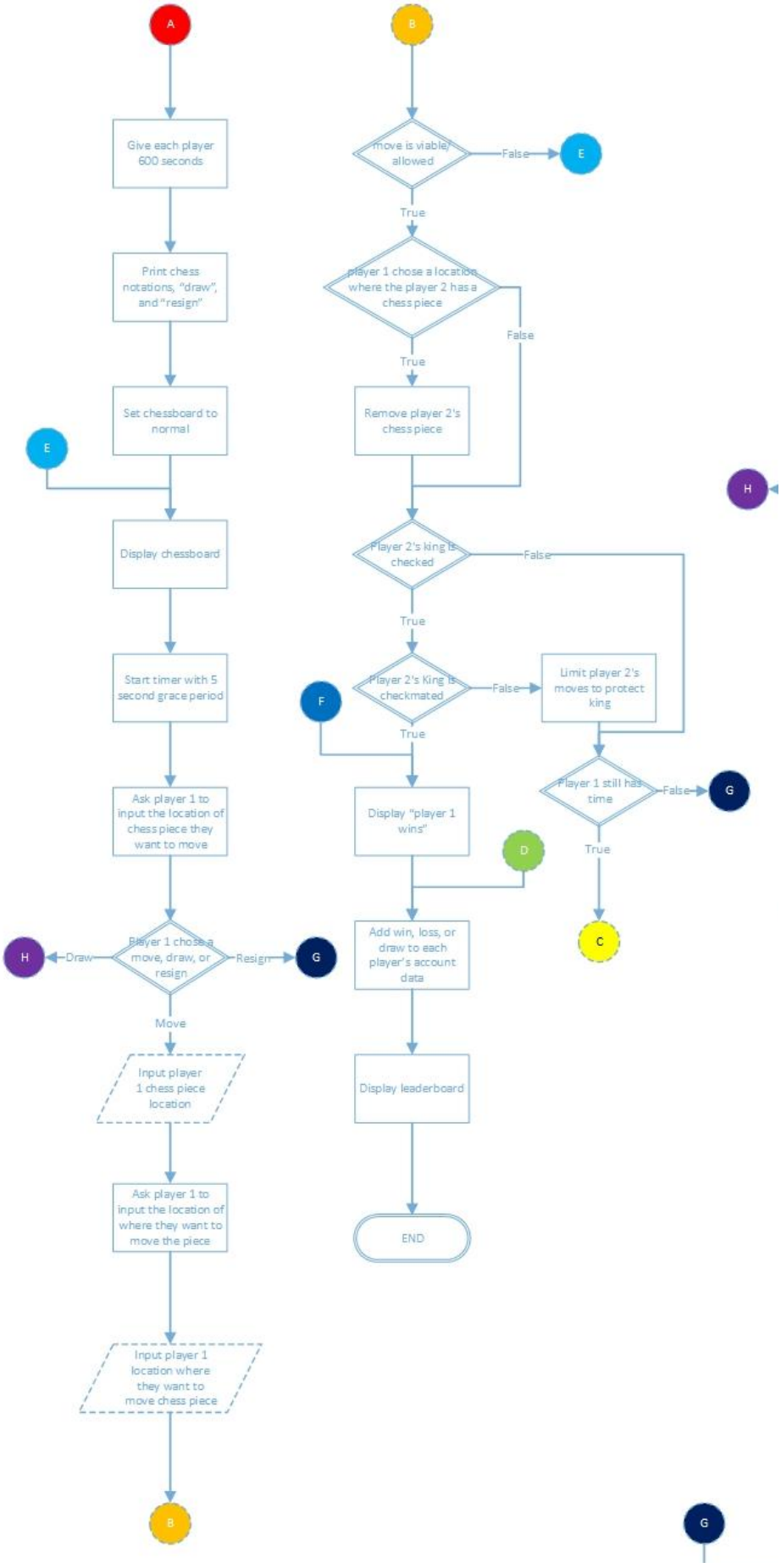
https://drive.google.com/file/d/1QWlco2-q-TVdqhnu5hO-i9la5cnyHWKM/view?usp=sharing

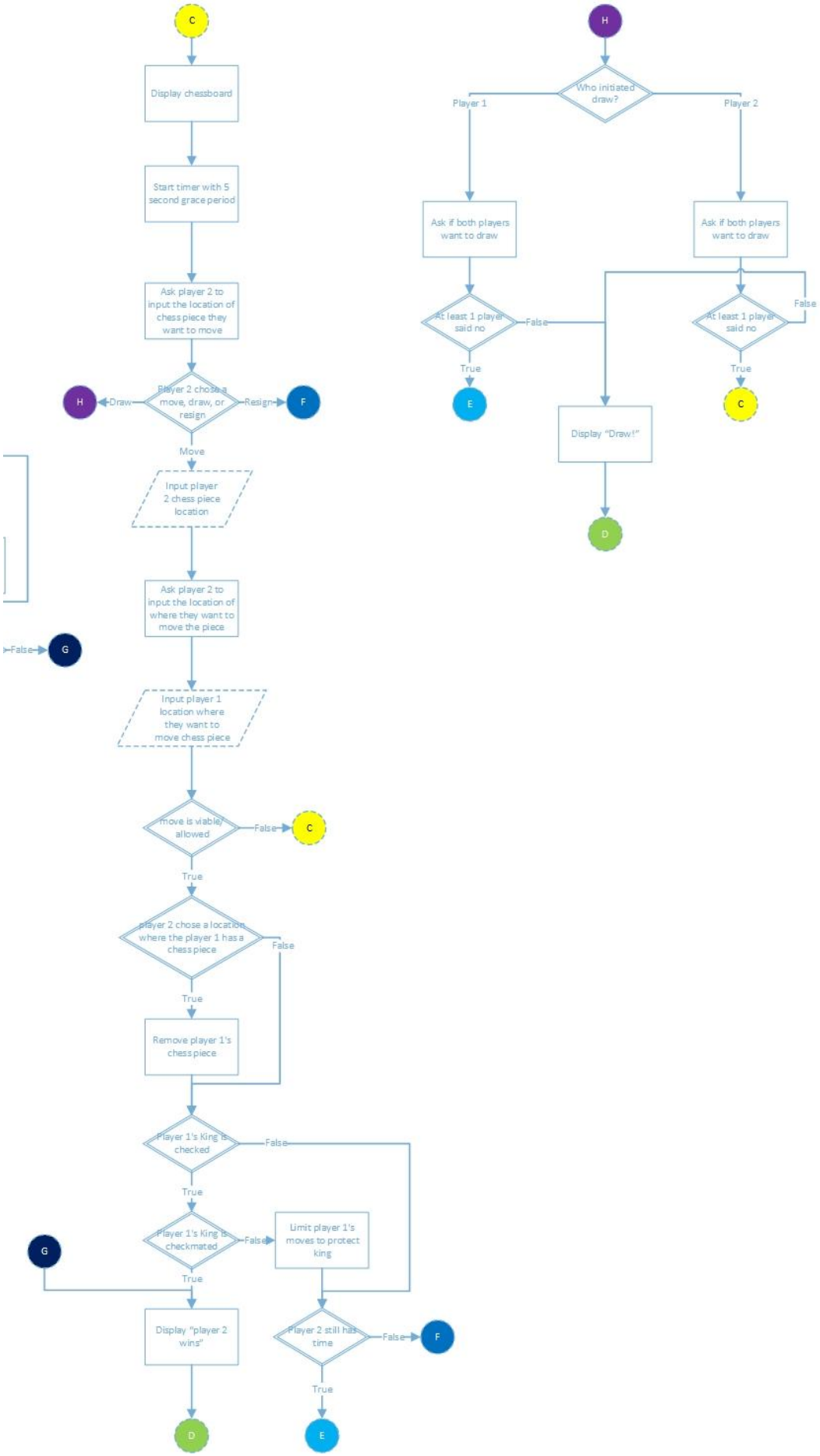


C. Flowchart

https://drive.google.com/file/d/1RkQG4a4fQ6U3a5lg2sQADHq2hD32fHpk/view?usp=share_link


START
Load player account database
Player account database
Ask player 1 or player 2 to login or signup?
User chose signup
False
Ask for login credentials
True
Ask for new username and password
Input login credentials
Input new username and password
Username is already in use
False
Both players done logging in/ signing up
False
True
Display "Username already in use, please input again."
True
A


A
Give each player 600 seconds
Print chess notations, "draw", and "resign"
Set chessboard to normal
E
Display chessboard
Start timer with 5 second grace period
Ask player 1 to input the location of chess piece they want to move
Player 1 chose a move, draw, or resign
H
Draw
Resign
G
Move
Input player 1 chess piece location
Ask player 1 to input the location of where they want to move the piece
Input player 1 location where they want to move chess piece
B
B
move is viable/ allowed
False
E
True
player 1 chose a location where the player 2 has a chess piece
False
True
Remove player 2's chess piece
H
Player 2's king is checked
False
True
Player 2's King is checkmated
False
Limit player 2's moves to protect king
F
True
Player 1 still has time
False
G
True
C
Display "player 1 wins"
D
Add win, loss, or draw to each player's account data
Display leaderboard
END
G


C
Display chessboard
Start timer with 5 second grace period
Ask player 2 to input the location of chess piece they want to move
Player 2 chose a move, draw, or resign
H
Draw
Resign
F
Move
Input player 2 chess piece location
Ask player 2 to input the location of where they want to move the piece
False
G
Input player 1 location where they want to move chess piece
move is viable/ allowed
False
C
True
player 2 chose a location where the player 1 has a chess piece
False
True
Remove player 1's chess piece
Player 1's King is checked
False
True
Player 1's King is checkmated
False
Limit player 1's moves to protect king
G
True
Display "player 2 wins"
Player 2 still has time
False
F
True
D
E
H
Who initiated draw?
Player 1
Player 2
Ask if both players want to draw
Ask if both players want to draw
At least 1 player said no
False
At least 1 player said no
False
True
True
E
C
Display "Draw!"
D

D. Pseudocode

```
Start

Declare all variables

Input Player1_username, Player1_password

If Player1 is logging in Then

    If Player1's information aligns with an account in the account database Then

        Go to Player2 login/signup

    Else

        Input Player1_username, Player1_password again

    Endif

Else If Player1 is signing up Then

    If Player1 username does not align with an account in the account database Then

        Go to Player2 login/signup

    Else

        Input Player1_username, Player1_password again

    Endif

Endif

Input Player2_username, Player2_password

If Player2 is logging in Then

    If Player2's information aligns with an account in the account database Then

        Go to Menu

    Else

        Input Player2_username, Player2_password again
```

```
        Endif
Else If Player2 is signing up Then
        If Player2 username does not align with an account in the account database Then
                Go to Menu
        Else
                Input Player2_username, Player2_password again
        Endif
Endif
Display list of possible moves
While game has not ended (A player's time has not reached 0 and both kings are on the board) Do
        Countdown time (initial time of 10 minutes, counts down per second while there is no input, 5 second grace period before start of countdown)
        For each player in [Player1, Player2] (Cycles between the two until game ends)
                Get move input from player using chess notation (may also draw or resign)
                If draw Then
                        Ask both players if they want to draw
                        If yes Then
                                End game and add draw to player stats
                        Else
                                Continue game
                        Endif
                Else If resign Then
```

End game and add loss to resigning player's stats and add win to other player's stats

Else

While move input is not valid Do

Get move input from player using chess notation (may also draw or resign) until valid

Endwhile

Do chess move

Endif

Update chessboard with player move inputs

Update time

Check if any game end scenarios have been reached

If yes Then

End game

Else

Continue game

Endif

Endfor

Display current time per player (updates and prints every second) and current chessboard (updates and prints after every move)

Endwhile

Display player stats, winner of chess game, and leaderboard according to account database wins.

End

IV. Results

Account Login and/or Signup:

```
Player 1 (White), login or signup? test

Invalid command. Input again.

Player 1 (White), login or signup? signup

Player 1 (White), input new account details:
Input New Username (20 chars): testacct
Input New Password (20 chars): ••••••••


Player 2 (Black), login or signup? login

Player 2 (Black), input account details:
Input Username (20 chars): joe
Input Password (20 chars): ••••
```

Main Menu:

```
Welcome, testacct and joe!

Menu:
   1. START
   2. LEADERBOARD
   3. END


Input Command: test

Invalid command, input again

Input Command: leaderboard



Leaderboard:

1. joe                      | Point Rating: 2.0

2. william                  | Point Rating: 2.0

3. Harold                   | Point Rating: 1.0

4. testacct                 | Point Rating: 0.0

5. ---                      | Point Rating: ---


Input Command: start
```

Move Input:

```
Possible moves:
 ['a1', 'a2', 'a3', 'a4', 'a5', 'a6', 'a7', 'a8', 'b1', 'b2', 'b3', 'b4', 'b5', 'b6', 'b7', 'b8', 'c1', 'c2', 'c3', 'c4', 'c5',
'c6', 'c7', 'c8', 'd1', 'd2', 'd3', 'd4', 'd5', 'd6', 'd7', 'd8', 'e1', 'e2', 'e3', 'e4', 'e5', 'e6', 'e7', 'e8', 'f1', 'f2',
'f3', 'f4', 'f5', 'f6', 'f7', 'f8', 'g1', 'g2', 'g3', 'g4', 'g5', 'g6', 'g7', 'g8', 'h1', 'h2', 'h3', 'h4', 'h5', 'h6', 'h7',
'h8', 'shortcastlewhite', 'longcastlewhite', 'shortcastleblack', 'longcastleblack', 'draw', 'resign']

*PS: For moves using normal chess notation (e.g. e2 to e4), please type them in the format 'move1,move2' (e.g. e2,e4).

testacct's Turn (White)
[['8' '♜' '♞' '♝' '♛' '♚' '♝' '♞' '♜']
 ['7' '♟' '♟' '♟' '♟' '♟' '♟' '♟' '♟' '♟']
 ['6' '  ' '  ' '  ' '  ' '  ' '  ' '  ' '  ']
 ['5' '  ' '  ' '  ' '  ' '  ' '  ' '  ' '  ']
 ['4' '  ' '  ' '  ' '  ' '  ' '  ' '  ' '  ']
 ['3' '  ' '  ' '  ' '  ' '  ' '  ' '  ' '  ']
 ['2' '♙' '♙' '♙' '♙' '♙' '♙' '♙' '♙']
 ['1' '♖' '♘' '♗' '♕' '♔' '♗' '♘' '♖']
 ['-' 'a' ' b' ' c' ' d' ' e' ' f' ' g' 'h']]
Input move: test
09:56
Invalid, input again
Input move: e2,e4
09:56
Move 1 : e2 to e4 (White)

joe's Turn (Black)
[['1' '♖' '♘' '♗' '♔' '♕' '♗' '♘' '♖']
 ['2' '♙' '♙' '♙' '♙' '  ' '♙' '♙' '♙' '♙']
 ['3' '  ' '  ' '  ' '  ' '  ' '  ' '  ' '  ']
 ['4' '  ' '  ' '  ' '  ' '♙' '  ' '  ' '  ' '  ']
 ['5' '  ' '  ' '  ' '  ' '  ' '  ' '  ' '  ']
 ['6' '  ' '  ' '  ' '  ' '  ' '  ' '  ' '  ']
 ['7' '♟' '♟' '♟' '♟' '♟' '♟' '♟' '♟']
 ['8' '♜' '♞' '♝' '♚' '♛' '♝' '♞' '♜']
 ['-' 'h' ' g' ' f' ' e' ' d' ' c' ' b' 'a']]
09:39

Input move:
```

Draw Offer:

```
joe's Turn (Black)
[['1' '♖' '♘' '♗' '♔' '♕' '♗' '♘' '♖']
 ['2' '♙' '♙' '♙' '♙' '  ' '♙' '♙' '♙' '♙']
 ['3' '  ' '  ' '  ' '  ' '  ' '  ' '  ' '  ']
 ['4' '  ' '  ' '  ' '  ' '♙' '  ' '  ' '  ' '  ']
 ['5' '  ' '  ' '  ' '  ' '  ' '  ' '  ' '  ']
 ['6' '  ' '  ' '  ' '  ' '  ' '  ' '  ' '  ']
 ['7' '♟' '♟' '♟' '♟' '♟' '♟' '♟' '♟']
 ['8' '♜' '♞' '♝' '♚' '♛' '♝' '♞' '♜']
 ['-' 'h' ' g' ' f' ' e' ' d' ' c' ' b' 'a']]
Input move: draw
07:40
Does Black confirm the draw (Y or N)?  y
Does White confirm the draw (Y or N)?  n
Continuing game...
07:40

Input move:
```

Castling:

```
testacct's Turn (White)
[['8' '♜' ' ' ' ' '♝' '♛' '♚' '♝' '♞' '♜']
 ['7' '♟' '♟' '♟' '♟' '♟' '♟' ' ' ' ' '♟']
 ['6' '♞' ' ' ' ' ' ' ' ' ' ' '♟' ' ' ' ']
 ['5' ' ' ' ' ' ' ' ' ' ' ' ' ' ' '♟' ' ']
 ['4' ' ' ' ' ' ' ' ' ' ' '♙' ' ' ' ' ' ']
 ['3' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' '♘']
 ['2' '♙' '♙' '♙' '♙' '♗' '♙' '♙' '♙' '♙']
 ['1' '♖' '♘' '♗' '♕' '♔' ' ' ' ' ' ' '♖']
 ['-' 'a' ' b' ' c' ' d' ' e' ' f' ' g' 'h']]
Input move: shortcastlewhite
09:26
Move 7 : short castle (White)

joe's Turn (Black)
[['1' ' ' ' ' '♔' '♖' ' ' ' ' '♕' '♗' '♘' '♖']
 ['2' '♙' '♙' '♙' '♙' '♗' '♙' '♙' '♙' '♙']
 ['3' '♘' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ']
 ['4' ' ' ' ' ' ' ' ' '♙' ' ' ' ' ' ' ' ']
 ['5' ' ' ' ' '♟' ' ' ' ' ' ' ' ' ' ' ' ']
 ['6' ' ' ' ' ' ' '♟' ' ' ' ' ' ' ' ' '♞']
 ['7' '♟' '♟' ' ' ' ' '♟' '♟' '♟' '♟' '♟']
 ['8' '♜' '♞' '♝' '♚' '♕' '♝' ' ' ' ' '♜']
 ['-' 'h' ' g' ' f' ' e' ' d' ' c' ' b' 'a']]
06:07

Input move:
```

Check:

```
testacct's Turn (White)
[['8' '♜' ' ' ' ' '♝' '♛' '♚' '♝' ' ' ' ' '♜']
 ['7' '♟' '♟' '♟' '♟' ' ' '♟' ' ' ' ' ' ' '♟']
 ['6' '♞' ' ' ' ' ' ' '♟' ' ' ' ' '♟' ' ' '♞']
 ['5' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' '♟' ' ' ']
 ['4' ' ' ' ' ' ' '♗' ' ' ' ' '♙' ' ' ' ' ' ']
 ['3' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' '♘']
 ['2' '♙' '♙' '♙' '♙' '♙' ' ' ' ' '♙' '♙' '♙']
 ['1' '♖' '♘' '♗' '♕' ' ' ' ' '♖' '♔' ' ' ' ']
 ['-' 'a' ' b' ' c' ' d' ' e' ' f' ' g' 'h']]
Input move: c4,b5
08:54
Move 11 : c4 to b5 (White)

joe's Turn (Black)
[['1' ' ' ' ' '♔' '♖' ' ' ' ' '♕' '♗' '♘' '♖']
 ['2' '♙' '♙' '♙' '♙' ' ' ' ' '♙' '♙' '♙' '♙']
 ['3' '♘' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ']
 ['4' ' ' ' ' ' ' ' ' '♙' ' ' ' ' ' ' ' ']
 ['5' ' ' ' ' '♟' ' ' ' ' ' ' ' ' '♗' ' ' ']
 ['6' '♞' ' ' ' ' '♟' ' ' ' ' '♟' ' ' ' ' '♞']
 ['7' '♟' '♟' ' ' ' ' ' ' '♟' ' ' '♟' '♟' '♟']
 ['8' '♜' ' ' ' ' '♗' '♚' '♕' '♝' ' ' ' ' '♜']
 ['-' 'h' ' g' ' f' ' e' ' d' ' c' ' b' 'a']]
Input move: a6,b4
04:34
Move leads to being checked, input again
04:27

Input move:
```

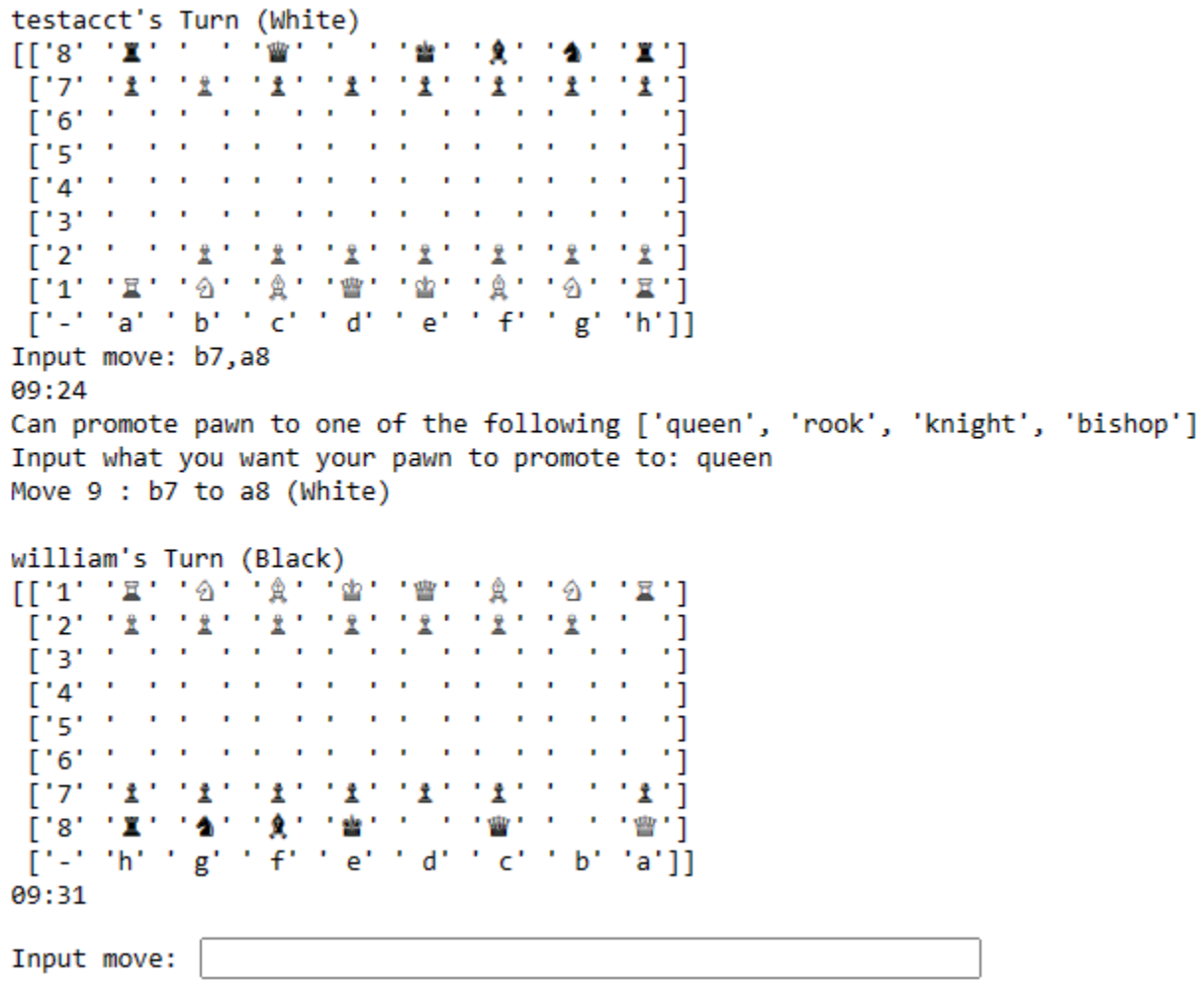
Checkmate, End Screen, & Leaderboard Update:

```
testacct's Turn (White)
[['8' '♜' ' ' ' ' ' ' '♕' '♚' '♝' ' ' ' ' '♜']
 ['7' '♟' '♟' '♟' '♝' '♟' ' ' ' ' ' ' '♟']
 ['6' ' ' ' ' ' ' ' ' '♟' ' ' ' ' '♟' ' ' '♞']
 ['5' ' ' ' '♗' ' ' ' ' ' ' ' ' ' ' '♟' ' ' ' ']
 ['4' ' ' ' '♞' ' ' ' ' ' ' '♙' ' ' ' ' ' ' ' ']
 ['3' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' '♘']
 ['2' '♙' '♙' '♙' '♙' ' ' ' ' '♙' '♙' '♙' '♙']
 ['1' '♖' '♘' '♗' '♕' ' ' ' ' '♖' '♔' ' ' ' ']
 ['-' 'a' ' b' ' c' ' d' ' e' ' f' ' g' 'h']]
Input move: d1,h5
08:47
Move 13 : d1 to h5 (White)

White wins! (Black's King was checkmated)



Leaderboard:

1. joe                          | Point Rating: 2.0

2. william                      | Point Rating: 2.0

3. Harold                       | Point Rating: 1.0

4. testacct                     | Point Rating: 1.0

5. ---                          | Point Rating: ---



Final Board:

[['8' '♜' ' ' ' ' ' ' '♕' '♚' '♝' ' ' ' ' '♜']
 ['7' '♟' '♟' '♟' '♝' '♟' ' ' ' ' ' ' '♟']
 ['6' ' ' ' ' ' ' ' ' '♟' ' ' ' ' '♟' ' ' '♞']
 ['5' ' ' ' '♗' ' ' ' ' ' ' ' ' ' ' '♟' ' ' '♕']
 ['4' ' ' ' '♞' ' ' ' ' ' ' '♙' ' ' ' ' ' ' ' ']
 ['3' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' '♘']
 ['2' '♙' '♙' '♙' '♙' ' ' ' ' '♙' '♙' '♙' '♙']
 ['1' '♖' '♘' '♗' ' ' ' ' ' ' '♖' '♔' ' ' ' ']
 ['-' 'a' ' b' ' c' ' d' ' e' ' f' ' g' 'h']]


-END-
```

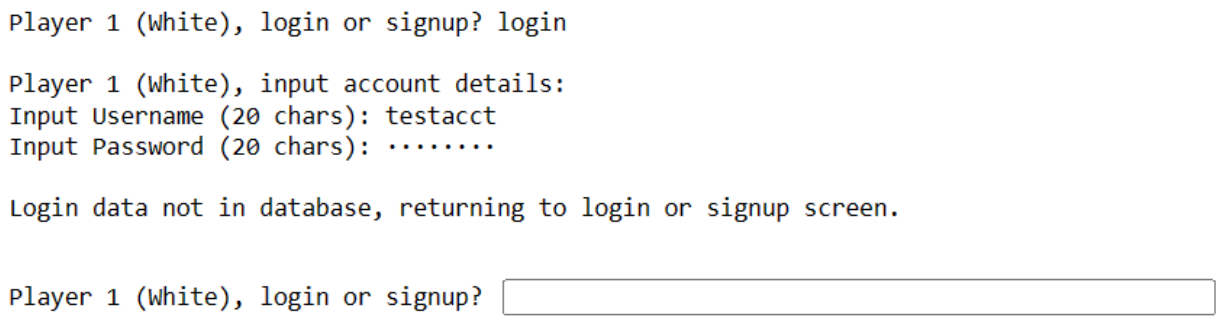
Resignation:

```
joe's Turn (Black)
[['1' '♖' '♘' '♗' '♔' '♕' '♗' '♘' '♖']
 ['2' '♙' '♙' '♙' '  ' '♙' '♙' '♙' '♙']
 ['3' '  ' '  ' '  ' '  ' '  ' '  ' '  ' '  ']
 ['4' '  ' '  ' '  ' '♙' '  ' '  ' '  ' '  ']
 ['5' '  ' '  ' '  ' '  ' '  ' '  ' '  ' '  ']
 ['6' '  ' '  ' '  ' '  ' '  ' '  ' '  ' '  ']
 ['7' '♟' '♟' '♟' '♟' '♟' '♟' '♟' '♟']
 ['8' '♜' '♞' '♝' '♚' '♛' '♝' '♞' '♜']
 ['-' 'h' ' g' ' f' ' e' ' d' ' c' ' b' 'a']]
Input move: resign
10:00

White wins! (Black has resigned)



Leaderboard:

1. joe                          | Point Rating: 2.0

2. william                      | Point Rating: 2.0

3. testacct                     | Point Rating: 2.0

4. Harold                       | Point Rating: 1.0

5. ---                          | Point Rating: ---


Final Board:

[['8' '♜' '♞' '♝' '♛' '♚' '♝' '♞' '♜']
 ['7' '♟' '♟' '♟' '♟' '♟' '♟' '♟' '♟']
 ['6' '  ' '  ' '  ' '  ' '  ' '  ' '  ' '  ']
 ['5' '  ' '  ' '  ' '  ' '  ' '  ' '  ' '  ']
 ['4' '  ' '  ' '  ' '  ' '♙' '  ' '  ' '  ']
 ['3' '  ' '  ' '  ' '  ' '  ' '  ' '  ' '  ']
 ['2' '♙' '♙' '♙' '♙' '  ' '♙' '♙' '♙']
 ['1' '♖' '♘' '♗' '♕' '♔' '♗' '♘' '♖']
 ['-' 'a' ' b' ' c' ' d' ' e' ' f' ' g' 'h']]


-END-
```

Pawn Promotion:



Wrong Password:



Username in Use:

```
Player 1 (White), login or signup? signup

Player 1 (White), input new account details:
Input New Username (20 chars): joe
Input New Password (20 chars): ••••••••

Username already in use, please input again.

Player 1 (White), input new account details:

Input New Username (20 chars):
```

End from Menu:

```
Player 2 (Black), login or signup? login

Player 2 (Black), input account details:
Input Username (20 chars): joe
Input Password (20 chars): ••••••••


Welcome, asdf and joe!

Menu:
   1. START
   2. LEADERBOARD
   3. END


Input Command: end



-END-
```

V. Discussion of Results

The developed chess game met most of the objectives and requirements that were initially set. The game's performance was tested using various test cases and gameplay scenarios, and the results showed that the game performed well and did not exhibit any significant latency or delays during gameplay. The game was optimized to handle complex operations in real-time while managing computational resources effectively. Moreover, most of the required functionalities such as castling, checkmate checker, pawn promotion, and double pawn move were implemented.When an invalid move is inputted, such as a knight moving diagonally, the program would inform the user that it is an invalid move and would ask for another input. However, due to time constraints, en passant was not implemented. Despite that, other features such as password encryption, user login and signup, and leaderboard with corresponding points were all implemented as well.

VI. Analysis, Conclusion and Future Directives

Overall, the results of this project have demonstrated that creating a chess game using Python can be successfully developed. The project was able to achieve most of the objectives that were set, providing a valuable learning opportunity that is open for further experimentation and development.

In conclusion, it can be safe to say that the resulting program is able to perform the deeds of the original proposal of an array-based Python program by utilizing the nature of CharArrays to demonstrate and utilize both the characteristics of arrays to be able to simulate location mapping and strings/unicode values to display and store chess piece data. The program was further improved with Quality of Life (QoL) features like including a countdown timer and password input protection through modules such as time and getpass. This allows the Python chess program to imitate and be as accurate as possible to the real life counterpart of standard chess. Although the program may still be able to improve in many aspects.

Some aspects the program can improve on in the future through updates to the code are as follows:

1. En Passant - The exclusion of En Passant was due to lacking time, En Passant may be able to be added in a future update given time. En Passant can be coded into the chess program by utilizing the chessmoves dictionary in the code to detect the state of the enemy pawns if they have moved just once and have done a double-step pawn move, as well as the location of the player's own pawns if they have just passed the middle trench of the chess board. The data gathered from those will be able to add the En Passant to the move list.
2. Board Coloring - Due to the nature of CharArrays to only store single strings or unicode values, it is not possible to put black or white squares at the bottom of the board if there is a piece in that specific array location. The board coloring will have to be made utilizing GUI designs which is something users of the program can look into in the future by adding GUI-based programming into the code.
3. Time Series Data - The program currently does not have the ability to undo moves, which some other chess based programs have. Although, it can be argued that the program is following the international chess rule of "Touch Move" since that is the case. Nonetheless, the inclusion of Time Series Data through storing the state of the board in each move using a dictionary is possible, allowing undos and the ability to see previous states of the board (although the program already technically does that by printing the state of the board and not clearing after every move).
4. Clearing the Console - Another feature that can be added is the clearing of the console after moves, the program can display only the two latest states of the board (one for white's last move and one for black's) which makes the console significantly less cluttered. This can probably be done with saving the state of the board by using the Time Series Data addition from the former paragraph or maybe through the use of end="\r" or the os module with os.system("clear") or os.system("cls").

References

Fiekas, N. (2022, December 22). *chess 1.9.4*. PyPI. Retrieved April 20, 2023 from https://pypi.org/project/chess/.

Gaddis, T. (2008). *Starting Out with Programming Logic and Design.* Pearson Education India.

MasterClass. (2022, September 3). *Chess 101: Who invented chess? learn about the history of chess and 3 memorable chess games - 2023*. MasterClass. Retrieved April 20, 2023 from https://www.masterclass.com/articles/chess-101-who-invented-chess-learn-about-the-history-of-chess-and-3-memorable-chess-games.

Mitev, L. (2011, August 9). *Simple Python Chess*. GitHub. Retrieved April 20, 2023 from https://github.com/liudmil-mitev/Simple-Python-Chess.

NumPy. (n.d.). *numpy.chararray*. numpy.org. Retrieved April 20, 2023 from https://numpy.org/doc/stable/reference/generated/numpy.chararray.html.

Pychess. (n.d.). *About pychess*. PyChess.org. Retrieved April 20, 2023 from https://www.pychess.org/about.

Python. (2023a). *queue — A synchronized queue class*. docs.python.org. Retrieved April 20, 2023 from https://docs.python.org/3/library/queue.html.

Python. (2023b). *threading — Thread-based parallelism*. docs.python.org. Retrieved April 20, 2023 from https://docs.python.org/3/library/threading.html.

Python. (2023c). *time — Time access and conversions*. docs.python.org. Retrieved April 20, 2023 from https://docs.python.org/3/library/time.html.

Rath, P. K., Mahapatro, N., Nath, P., & Dash, R. (2019, October). Autonomous Chess Playing Robot. In *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)* (pp. 1-6). IEEE. Retrieved April 20, 2023 from https://www.researchgate.net/profile/Prabin-Rath/publication/338590711_Autonomous_

Chess_Playing_Robot/links/5e345d33a6fdccd9657bf504/Autonomous-Chess-Playing-Robot.pdf.

Ruiz, R., & Pakzad, A. (2023). *Arrays - PROLOGI lecture 8 part 1*. DLSU Instructure. Retrieved April 20, 2023 from https://dlsu.instructure.com/courses/121965/files/folder/lesson?preview=13592519.

Satyam, K. (2020, December 23). *Python getpass module*. GeeksforGeeks. Retrieved April 20, 2023 from https://www.geeksforgeeks.org/python-getpass-module/.

Vallance, L. (2018, April 19). *Exploring the Python Chess Module.* liamvallance.com. Retrieved April 20, 2023 from http://liamvallance.com/img/Exploring%20python%20chess.pdf.

W3Schools. (n.d.). *Python Strings Are Arrays*. w3schools.com. Retrieved April 20, 2023 from https://www.w3schools.com/python/gloss_python_strings_are_arrays.asp.

Appendices

A. User's Manual

For login and signup:

Type in "login" or "signup" in the input box to login or signup, any other text will be rejected, capitalization does not matter but whitespace does (i.e. "LoGiN" is accepted but " login" is not).

For username and password:

Username and password input only accepts up to 20 alphanumeric characters, input cannot be empty.

For move input:

For normal chess moves like e2 to e4, input should be in the style "e2,e4", or basically separate the initial location and the final location with a comma.

For other moves like castling, promotion, draw offer, and resignation, simply type the move that corresponds to it in the possible moves list that will be printed (i.e. resignation is “resign”).

B. Source Code –

```
from numpy import chararray #Only importing chararray as numpy and queue both have put() functions
import time
import threading as thr
import queue as que
import getpass as gp


def startchess():


    user1, user2 = login() #Fetch account names


    startinput = ""
    startcommands = ["start","leaderboard","end","1","2","3"]
    print(f"Menu: \n  1. {startcommands[0].upper()} \n  2. {startcommands[1].upper()} \n  3. {startcommands[2].upper()} \n")
    while True:
        while startinput not in startcommands:
            startinput = str(input("\nInput Command: ")).lower()
            print("")
            if startinput not in startcommands:
                print("Invalid command, input again")
        if startinput == "start" or startinput == "1":
            break
        if startinput == "leaderboard" or startinput == "2":
```

```
        updateleaderboard(user1, 0, 0, 0, user2, 0, 0, 0)  #Fetch leaderboard by updating values by 0 (No change)
        startinput = ""
    if startinput == "end" or startinput == "3":
        print("\n")
        return "-END-"


  chesslist = ["\u265C","\u265E","\u265D","\u265B","\u265A","\u265D","\u265E","\u265C",
               "\u265F","\u265F","\u265F","\u265F","\u265F","\u265F","\u265F","\u265F",
               "\u3164","\u3164","\u3164","\u3164","\u3164","\u3164","\u3164","\u3164",
               "\u3164","\u3164","\u3164","\u3164","\u3164","\u3164","\u3164","\u3164",
               "\u3164","\u3164","\u3164","\u3164","\u3164","\u3164","\u3164","\u3164",
               "\u3164","\u3164","\u3164","\u3164","\u3164","\u3164","\u3164","\u3164",
               "\u2659","\u2659","\u2659","\u2659","\u2659","\u2659","\u2659","\u2659",
               "\u2656","\u2658","\u2657","\u2655","\u2654","\u2657","\u2658","\u2656"]  #Chess pieces in Unicode
  chesslistlegacy = chesslist  #Back-up board in case of need for board reversion
  notation = ["a1","a2","a3","a4","a5","a6","a7","a8","b1","b2","b3","b4","b5","b6","b7","b8",
              "c1","c2","c3","c4","c5","c6","c7","c8","d1","d2","d3","d4","d5","d6","d7","d8",
              "e1","e2","e3","e4","e5","e6","e7","e8","f1","f2","f3","f4","f5","f6","f7","f8",
              "g1","g2","g3","g4","g5","g6","g7","g8","h1","h2","h3","h4","h5","h6","h7","h8",
              "shortcastlewhite","longcastlewhite","shortcastleblack","longcastleblack",
              "draw","resign"]  #Chess notation list


  chessmoves = {} #Dictionary to check for initial moves (for check and two-step pawn), counts how many times a tile has been played on
  for x in range(64):
      chessmoves[notation[x]] = 0
```

```
    chesstimew = 600  #Initial time of ten minutes per player
    chesstimeb = 600
    count = 1     #Move count
    print("Possible moves: \n", notation)
    print("\n*PS: For moves using normal chess notation (e.g. e2 to e4), please type them in the format 'move1,move2' (e.g. e2,e4).")
    time.sleep(1)


    while True:
        #White's move
        if count%2 == 1:
            print(f"\n{user1}'s Turn (White)")
            print(board(chesslist))
            imove, fmove = " ", " "
            while imove not in notation or fmove not in notation:
                chesstime = chesstimew
                move = thr.Event()
                remtime = que.Queue()
                result = que.Queue()
                timerthread = thr.Thread(target = timer, args = (remtime, chesstime, move))
                movethread = thr.Thread(target = moveinput, args = (result, move,))
                #Making thread to countdown the time while waiting for move input
                timerthread.start()
                movethread.start()


                movethread.join()
                timerthread.join()
                #End thread and join with main program
                moves = result.get()  #Get move input from thread
```

```
chesstimew = remtime.get()  #Get remaining time from thread
if "," in moves:
    moves = moves.split(",")
    moves = tuple([moves[0], moves[1]])
else:
    moves = tuple([moves, moves])
imove = moves[0].lower()
fmove = moves[1].lower()

if matecheck(chesslist, board(chesslist), color="white")[2] == 0:
    validinitial = notation
    validfinal = notation
if matecheck(chesslist, board(chesslist), color="white")[2] == 1:
    validinitial = notation[:64]+notation[-2:]
    validfinal = notation[:64]+notation[-2:]
if imove == None and fmove == None:
    print("Input move")
elif imove not in validinitial or fmove not in validfinal:
    print("Invalid, input again")
    imove, fmove = " ", " "
elif imove == "resign" and fmove == "resign":
    print("\nBlack wins! (White has resigned)")
    updateleaderboard(user1, 0, 1, 0, user2, 1, 0, 0)
    print("\nFinal Board:\n")
    print(board(chesslist))
    print("\n")
    return "-END-"
elif imove == "draw" and fmove == "draw":
```

```
    confirmw = input("Does White confirm the draw (Y or N)?  ").lower()
    confirmb = input("Does Black confirm the draw (Y or N)?  ").lower()
    if confirmw == "y" and confirmb == "y":
        print("\nDraw!")
        updateleaderboard(user1, 0, 0, 1, user2, 0, 0, 1)
        print("\nFinal Board:\n")
        print(board(chesslist))
        print("\n")
        return "-END-"
    else:
        print("Continuing game...")
        imove, fmove = " ", " "
elif castling(imove, fmove, chesslist, chessmoves, color="white")[1] == 1:
    chesslist = castling(imove, fmove, chesslist, chessmoves, color="white")[0]
    if matecheck(chesslist, board(chesslist), color="white")[2] != 0:
        print("Move leads to being checked, input again")
        chesslist = chesslistlegacy
        imove, fmove = " ", " "
    else:
        chessmoves["e1"]+=1
        if imove not in notation[64:68]:
            imove = fmove
        if fmove not in notation[64:68]:
            fmove = imove
        if imove == "shortcastlewhite" or fmove == "shortcastlewhite":
            chessmoves["h1"]+=1
        elif imove == "longcastlewhite" or fmove == "longcastlewhite":
            chessmoves["a1"]+=1
```

```
                chesslistlegacy = chesslist
                break
            elif chessmovew(imove, fmove, board(chesslist), chessmoves) == 1:
                chesslist[listcheck(fmove)], chesslist[listcheck(imove)] =
chesslist[listcheck(imove)], "\u3164"
                if matecheck(chesslist, board(chesslist), color="white")[2] != 0:
                    print("Move leads to being checked, input again")
                    chesslist = chesslistlegacy
                    imove, fmove = " ", " "
                else:
                    chessmoves[imove]+=1
                    chessmoves[fmove]+=1
                    chesslistlegacy = chesslist
                    break
            else:
                print("Invalid move, input again")
                imove, fmove = " ", " "
        chesslist = promotion(chesslist)
        if imove in notation[:64] and fmove in notation[:64]:
            print("Move",count,":",imove,"to",fmove,"(White)")
        elif imove == "shortcastlewhite" or fmove == "shortcastlewhite":
            print("Move",count,": short castle (White)")
        elif imove == "longcastlewhite" or fmove == "longcastlewhite":
            print("Move",count,": long castle (White)")
        if matecheck(chesslist, board(chesslist), color="black")[2] == -1:
            print("\nWhite wins! (Black's King was checkmated)")
            updateleaderboard(user1, 1, 0, 0, user2, 0, 1, 0)
            break
        if chesstimew == 0:
```

```
        print("\nBlack wins! (White ran out of time)")
        updateleaderboard(user1, 0, 1, 0, user2, 1, 0, 0)
        break
    if "\u265A" not in chesslist:
        print("\nWhite wins! (Black's King was captured)")
        updateleaderboard(user1, 1, 0, 0, user2, 0, 1, 0)
        break
    count+=1

#Black's move
if count%2 == 0:
    print(f"\n{user2}'s Turn (Black)")
    print(board(chesslist, "black"))
    imove, fmove = " ", " "
    while imove not in notation or fmove not in notation:
        chesstime = chesstimeb
        move = thr.Event()
        remtime = que.Queue()
        result = que.Queue()
        timerthread = thr.Thread(target = timer, args = (remtime, chesstime, move))
        movethread = thr.Thread(target = moveinput, args = (result, move,))
        #Making thread to countdown the time while waiting for move input
        timerthread.start()
        movethread.start()

        movethread.join()
        timerthread.join()
        #End thread and join with main program
```

```
moves = result.get()  #Get move input from thread
chesstimeb = remtime.get()  #Get remaining time from thread
if "," in moves:
    moves = moves.split(",")
    moves = tuple([moves[0], moves[1]])
else:
    moves = tuple([moves, moves])
imove = moves[0].lower()
fmove = moves[1].lower()

if matecheck(chesslist, board(chesslist), color="black")[2] == 0:
    validinitial = notation
    validfinal = notation
if matecheck(chesslist, board(chesslist), color="black")[2] == 1:
    validinitial = notation[:64]+notation[-2:]
    validfinal = notation[:64]+notation[-2:]
if imove == None and fmove == None:
    print("Input move")
elif imove not in validinitial or fmove not in validfinal:
    print("Invalid, input again")
    imove, fmove = " ", " "
elif imove == "resign" or fmove == "resign":
    print("\nWhite wins! (Black has resigned)\n")
    updateleaderboard(user1, 1, 0, 0, user2, 0, 1, 0)
    print("\nFinal Board:\n")
    print(board(chesslist))
    print("\n")
    return "-END-"
```

```
elif imove == "draw" or fmove == "draw":
    confirmb = input("Does Black confirm the draw (Y or N)?  ").lower()
    confirmw = input("Does White confirm the draw (Y or N)?  ").lower()
    if confirmb == "y" and confirmw == "y":
        print("\nDraw!")
        updateleaderboard(user1, 0, 0, 1, user2, 0, 0, 1)
        print("\nFinal Board:\n")
        print(board(chesslist))
        print("\n")
        return "-END-"
    else:
        print("Continuing game...")
        imove, fmove = " ", " "
elif castling(imove, fmove, chesslist, chessmoves, color="black")[1] == 1:
    chesslist = castling(imove, fmove, chesslist, chessmoves, color="black")[0]
    if matecheck(chesslist, board(chesslist), color="black")[2] != 0:
        print("Move leads to being checked, input again")
        chesslist = chesslistlegacy
        imove, fmove = " ", " "
    else:
        chessmoves["e8"]+=1
        if imove not in notation[64:68]:
            imove = fmove
        if fmove not in notation[64:68]:
            fmove = imove
        if imove == "shortcastleblack" or fmove == "shortcastleblack":
            chessmoves["h8"]+=1
        elif imove == "longcastleblack" or fmove == "longcastleblack":
```

```
                    chessmoves["a8"]+=1
                chesslistlegacy = chesslist
                break
            elif chessmoveb(imove, fmove, board(chesslist), chessmoves) == 1:
                chesslist[listcheck(fmove)], chesslist[listcheck(imove)] =
chesslist[listcheck(imove)], "\u3164"
                if matecheck(chesslist, board(chesslist), color="black")[2] != 0:
                    print("Move leads to being checked, input again")
                    chesslist = chesslistlegacy
                    imove, fmove = " ", " "
                else:
                    chessmoves[imove]+=1
                    chessmoves[fmove]+=1
                    chesslistlegacy = chesslist
                    break
            else:
                print("Invalid move, input again")
                imove, fmove = " ", " "
        chesslist = promotion(chesslist)
        if imove in notation[:64] and fmove in notation[:64]:
            print("Move",count,":",imove,"to",fmove,"(Black)")
        elif imove == "shortcastleblack" or fmove == "shortcastleblack":
            print("Move",count,": short castle (Black)")
        elif imove == "longcastleblack" or fmove == "longcastleblack":
            print("Move",count,": long castle (Black)")
        if matecheck(chesslist, board(chesslist), color="white")[2] == -1:
            print("\nBlack wins! (White's King was checkmated)")
            updateleaderboard(user1, 0, 1, 0, user2, 1, 0, 0)
            break
```

```
        if chesstimeb == 0:
            print("\nWhite wins! (Black ran out of time)")
            updateleaderboard(user1, 1, 0, 0, user2, 0, 1, 0)
            break
        if "\u2654" not in chesslist:
            print("\nBlack wins! (White's King was captured)")
            updateleaderboard(user1, 0, 1, 0, user2, 1, 0, 0)
            break
        count+=1


    print("\nFinal Board:\n")  #Print final state of the chess board once game is concluded
    print(board(chesslist))
    print("\n")
    return "-END-"


def chessmovew(initial, final, cbt, dct): #Move checker for white pieces, uses vectors (x,y) to find valid locations
    white = ["\u2656","\u2658","\u2657","\u2655","\u2654","\u2659"]
    chessnotation =
["a1","a2","a3","a4","a5","a6","a7","a8","b1","b2","b3","b4","b5","b6","b7","b8",
                "c1","c2","c3","c4","c5","c6","c7","c8","d1","d2","d3","d4","d5","d6","d7","d8",
                "e1","e2","e3","e4","e5","e6","e7","e8","f1","f2","f3","f4","f5","f6","f7","f8",
                "g1","g2","g3","g4","g5","g6","g7","g8","h1","h2","h3","h4","h5","h6","h7","h8"]
    intloc = location(initial)
    finloc = location(final)
    movelist = []
    if initial not in chessnotation or final not in chessnotation:
        return 0
    if cbt[intloc[0],intloc[1]] == "\u3164":
```

```
    print("Initial location does not contain chesspiece")
    return 0
if cbt[intloc[0],intloc[1]] not in white:
    print("Initial location contains enemy piece")
    return 0
if initial == final:
    print("Cannot move to same locations")
    return 0
if cbt[finloc[0],finloc[1]] in white:
    print("Move attacks own piece")
    return 0
if cbt[intloc[0],intloc[1]] == "\u2656":
    for xy in [(0,1),(1,0),(0,-1),(-1,0)]:
        x = intloc[0]
        y = intloc[1]
        while True:
            x += xy[0]
            y += xy[1]
            if not (0<=x<=7 and 1<=y<=8):
                break
            if cbt[x,y] == "\u3164":
                movelist.append(location((x,y)))
            elif cbt[x,y] not in white:
                movelist.append(location((x,y)))
                break
            else:
                break
if cbt[intloc[0],intloc[1]] == "\u2658":
```

```
    for xy in [(1,2),(2,1),(-1,2),(-2,1),(1,-2),(2,-1),(-1,-2),(-2,-1)]:
        x = intloc[0]
        y = intloc[1]
        while True:
            x += xy[0]
            y += xy[1]
            if not (0<=x<=7 and 1<=y<=8):
                break
            if cbt[x,y] not in white:
                movelist.append(location((x,y)))
                break
            else:
                break
if cbt[intloc[0],intloc[1]] == "\u2657":
    for xy in [(1,1),(-1,1),(1,-1),(-1,-1)]:
        x = intloc[0]
        y = intloc[1]
        while True:
            x += xy[0]
            y += xy[1]
            if not (0<=x<=7 and 1<=y<=8):
                break
            if cbt[x,y] == "\u3164":
                movelist.append(location((x,y)))
            elif cbt[x,y] not in white:
                movelist.append(location((x,y)))
                break
            else:
```

```
                break
if cbt[intloc[0],intloc[1]] == "\u2655":
    for xy in [(1,1),(-1,1),(1,-1),(-1,-1),(0,1),(1,0),(0,-1),(-1,0)]:
        x = intloc[0]
        y = intloc[1]
        while True:
            x += xy[0]
            y += xy[1]
            if not (0<=x<=7 and 1<=y<=8):
                break
            if cbt[x,y] == "\u3164":
                movelist.append(location((x,y)))
            elif cbt[x,y] not in white:
                movelist.append(location((x,y)))
                break
            else:
                break
if cbt[intloc[0],intloc[1]] == "\u2654":
    for xy in [(1,1),(-1,1),(1,-1),(-1,-1),(0,1),(1,0),(0,-1),(-1,0)]:
        x = intloc[0]
        y = intloc[1]
        while True:
            x += xy[0]
            y += xy[1]
            if not (0<=x<=7 and 1<=y<=8):
                break
            if cbt[x,y] not in white:
                movelist.append(location((x,y)))
```

```
                break
            else:
                break
    if cbt[intloc[0],intloc[1]] == "\u2659":
        x = intloc[0]
        y = intloc[1]
        if (0<=(x-1)<=7 and 1<=y<=8) and (cbt[x-1,y] == "\u3164"):
            movelist.append(location((x-1,y)))
        if (0<=(x-1)<=7 and 1<=(y-1)<=8) and (cbt[x-1,y-1] not in white+["\u3164"]):
            movelist.append(location((x-1,y-1)))
        if (0<=(x-1)<=7 and 1<=(y+1)<=8) and (cbt[x-1,y+1] not in white+["\u3164"]):
            movelist.append(location((x-1,y+1)))
        if (0<=(x-2)<=7 and 1<=y<=8) and (dct[initial] == 0) and (cbt[x-2,y] == "\u3164"):
            movelist.append(location((x-2,y)))
    if final in movelist:
        return 1
    else:
        return 0


def chessmoveb(initial, final, cbt, dct):  #Move checker for black pieces, uses vectors (x,y) to find valid locations
    black = ["\u265C","\u265E","\u265D","\u265B","\u265A","\u265F"]
    chessnotation =
["a1","a2","a3","a4","a5","a6","a7","a8","b1","b2","b3","b4","b5","b6","b7","b8",
                "c1","c2","c3","c4","c5","c6","c7","c8","d1","d2","d3","d4","d5","d6","d7","d8",
                "e1","e2","e3","e4","e5","e6","e7","e8","f1","f2","f3","f4","f5","f6","f7","f8",
                "g1","g2","g3","g4","g5","g6","g7","g8","h1","h2","h3","h4","h5","h6","h7","h8"]
    intloc = location(initial)
    finloc = location(final)
```

```
movelist = []
if initial not in chessnotation or final not in chessnotation:
    return 0
if cbt[intloc[0],intloc[1]] == "\u3164":
    print("Initial location does not contain chesspiece")
    return 0
if cbt[intloc[0],intloc[1]] not in black:
    print("Initial location contains enemy piece")
    return 0
if initial == final:
    print("Cannot move to same locations")
    return 0
if cbt[finloc[0],finloc[1]] in black:
    print("Move attacks own piece")
    return 0
if cbt[intloc[0],intloc[1]] == "\u265C":
    for xy in [(0,1),(1,0),(0,-1),(-1,0)]:
        x = intloc[0]
        y = intloc[1]
        while True:
            x += xy[0]
            y += xy[1]
            if not (0<=x<=7 and 1<=y<=8):
                break
            if cbt[x,y] == "\u3164":
                movelist.append(location((x,y)))
            elif cbt[x,y] not in black:
                movelist.append(location((x,y)))
```

```
            break
        else:
            break
if cbt[intloc[0],intloc[1]] == "\u265E":
    for xy in [(1,2),(2,1),(-1,2),(-2,1),(1,-2),(2,-1),(-1,-2),(-2,-1)]:
        x = intloc[0]
        y = intloc[1]
        while True:
            x += xy[0]
            y += xy[1]
            if not (0<=x<=7 and 1<=y<=8):
                break
            if cbt[x,y] not in black:
                movelist.append(location((x,y)))
                break
            else:
                break
if cbt[intloc[0],intloc[1]] == "\u265D":
    for xy in [(1,1),(-1,1),(1,-1),(-1,-1)]:
        x = intloc[0]
        y = intloc[1]
        while True:
            x += xy[0]
            y += xy[1]
            if not (0<=x<=7 and 1<=y<=8):
                break
            if cbt[x,y] == "\u3164":
                movelist.append(location((x,y)))
```

```
            elif cbt[x,y] not in black:
                movelist.append(location((x,y)))
                break
            else:
                break
    if cbt[intloc[0],intloc[1]] == "\u265B":
        for xy in [(1,1),(-1,1),(1,-1),(-1,-1),(0,1),(1,0),(0,-1),(-1,0)]:
            x = intloc[0]
            y = intloc[1]
            while True:
                x += xy[0]
                y += xy[1]
                if not (0<=x<=7 and 1<=y<=8):
                    break
                if cbt[x,y] == "\u3164":
                    movelist.append(location((x,y)))
                elif cbt[x,y] not in black:
                    movelist.append(location((x,y)))
                    break
                else:
                    break
    if cbt[intloc[0],intloc[1]] == "\u265A":
        for xy in [(1,1),(-1,1),(1,-1),(-1,-1),(0,1),(1,0),(0,-1),(-1,0)]:
            x = intloc[0]
            y = intloc[1]
            while True:
                x += xy[0]
                y += xy[1]
```

```
            if not (0<=x<=7 and 1<=y<=8):
                break
            if cbt[x,y] not in black:
                movelist.append(location((x,y)))
                break
            else:
                break
    if cbt[intloc[0],intloc[1]] == "\u265F":
        x = intloc[0]
        y = intloc[1]
        if (0<=(x+1)<=7 and 1<=y<=8) and (cbt[x+1,y] == "\u3164"):
            movelist.append(location((x+1,y)))
        if (0<=(x+1)<=7 and 1<=(y-1)<=8) and (cbt[x+1,y-1] not in black+["\u3164"]):
            movelist.append(location((x+1,y-1)))
        if (0<=(x+1)<=7 and 1<=(y+1)<=8) and (cbt[x+1,y+1] not in black+["\u3164"]):
            movelist.append(location((x+1,y+1)))
        if (0<=(x+2)<=7 and 1<=y<=8) and (cbt[x+2,y] == "\u3164") and (dct[initial] == 0):
            movelist.append(location((x+2,y)))
    if final in movelist:
        return 1
    else:
        return 0


def castling(initial, final, clt, dct, color):  #To check if castling is valid if no pieces between rook and king, and if they haven't been previously played yet
    casclt = clt
    castlelist = ["shortcastlewhite","longcastlewhite","shortcastleblack","longcastleblack"]
    if initial not in castlelist and final not in castlelist:
        return casclt, 0
```

```
    if color == "white":

        if (initial == "shortcastlewhite" and final == "longcastlewhite") or (initial ==
"longcastlewhite" and final == "shortcastlewhite"):

            print("Invalid move, input only one type of castle move")

            return casclt, 0

        elif (initial == "shortcastlewhite" or final == "shortcastlewhite") and (dct["e1"] == 0) and
(dct["h1"] == 0) and (casclt[listcheck("f1")] == "\u3164") and (casclt[listcheck("g1")] ==
"\u3164"):

            casclt[listcheck("e1")], casclt[listcheck("f1")], casclt[listcheck("g1")],
casclt[listcheck("h1")] = casclt[listcheck("f1")], casclt[listcheck("h1")], casclt[listcheck("e1")],
casclt[listcheck("g1")]

            return casclt, 1

        elif (initial == "longcastlewhite" or final == "longcastlewhite") and (dct["e1"] == 0) and
(dct["a1"] == 0) and (casclt[listcheck("b1")] == "\u3164") and (casclt[listcheck("c1")] ==
"\u3164") and (casclt[listcheck("d1")] == "\u3164"):

            casclt[listcheck("a1")], casclt[listcheck("b1")], casclt[listcheck("c1")],
casclt[listcheck("d1")], casclt[listcheck("e1")] = casclt[listcheck("d1")], casclt[listcheck("b1")],
casclt[listcheck("e1")], casclt[listcheck("a1")], casclt[listcheck("c1")]

            return casclt, 1

        else:

            return casclt, 0

    if color == "black":

        if (initial == "shortcastleblack" and final == "longcastleblack") or (initial ==
"longcastleblack" and final == "shortcastleblack"):

            print("Invalid move, input only one type of castle move")

            return casclt, 0

        elif (initial == "shortcastleblack" or final == "shortcastleblack") and dct["e8"] == 0 and
dct["h8"] == 0 and casclt[listcheck("f8")] == "\u3164" and casclt[listcheck("g8")] == "\u3164":

            casclt[listcheck("e8")], casclt[listcheck("f8")], casclt[listcheck("g8")],
casclt[listcheck("h8")] = casclt[listcheck("f8")], casclt[listcheck("h8")], casclt[listcheck("e8")],
casclt[listcheck("g8")]

            return casclt, 1
```

```
        elif (initial == "longcastleblack" or final == "longcastleblack") and dct["e8"] == 0 and dct["a8"] == 0 and casclt[listcheck("b8")] == "\u3164" and casclt[listcheck("c8")] == "\u3164" and casclt[listcheck("d8")] == "\u3164":

            casclt[listcheck("a8")], casclt[listcheck("b8")], casclt[listcheck("c8")], casclt[listcheck("d8")], casclt[listcheck("e8")] = casclt[listcheck("d8")], casclt[listcheck("b8")], casclt[listcheck("e8")], casclt[listcheck("a8")], casclt[listcheck("c8")]

            return casclt, 1

        else:

            return casclt, 0


def promotion(clt):  #To check if pawn reached opposite of board, and if so change pawn w/ better piece

    proclt = clt

    prolist = ["queen", "rook", "bishop", "knight"]

    whitepawnpro = ["a8","b8","c8","d8","e8","f8","g8","h8"]

    blackpawnpro = ["a1","b1","c1","d1","e1","f1","g1","h1"]

    promotion = ""

    for i in whitepawnpro:

        if proclt[listcheck(i)] == "\u2659":

            print("Can promote pawn to one of the following ['queen', 'rook', 'knight', 'bishop']")

            while promotion not in prolist:

                promotion = input("Input what you want your pawn to promote to: ").lower()

            if promotion == "queen":

                proclt[listcheck(i)] = "\u2655"

            if promotion == "rook":

                proclt[listcheck(i)] = "\u2656"

            if promotion == "bishop":

                proclt[listcheck(i)] = "\u2657"

            if promotion == "knight":

                proclt[listcheck(i)] = "\u2658"
```

```
    for i in blackpawnpro:
        if proclt[listcheck(i)] == "\u265F":
            print("Can promote pawn to one of the following ['queen', 'rook', 'knight', 'bishop']")
            while promotion not in prolist:
                promotion = input("Input what you want your pawn to promote to: ").lower()
            if promotion == "queen":
                proclt[listcheck(i)] = "\u265B"
            if promotion == "rook":
                proclt[listcheck(i)] = "\u265C"
            if promotion == "bishop":
                proclt[listcheck(i)] = "\u265D"
            if promotion == "knight":
                proclt[listcheck(i)] = "\u265E"
    return proclt


def matecheck(clt, cbt, color):  #To check if King is safe, checked, or checkmated; This is done by checking if current space is being attacked, and checking if spaces where the king can move to are being attacked
    white = ["\u2656","\u2658","\u2657","\u2655","\u2654","\u2659"]
    black = ["\u265C","\u265E","\u265D","\u265B","\u265A","\u265F"]
    if color == "white":
        for i in range(len(clt)):
            if clt[i] == "\u2654":
                king = i
                break
        kingstart = listcheck(king)
        king = location(kingstart)
        movelist = [(king[0],king[1])]
        for xy in [(1,1),(-1,1),(1,-1),(-1,-1),(0,1),(1,0),(0,-1),(-1,0)]:
```

```
    x = king[0]
    y = king[1]
    while True:
        x += xy[0]
        y += xy[1]
        if not (0<=x<=7 and 1<=y<=8):
            break
        if cbt[x,y] not in white:
            movelist.append((x,y))
            break
        else:
            break
checkcount = 0
for xy in [(0,1),(1,0),(0,-1),(-1,0)]:
    x = movelist[0][0]
    y = movelist[0][1]
    while True:
        x += xy[0]
        y += xy[1]
        if not (0<=x<=7 and 1<=y<=8):
            break
        if cbt[x,y] == "\u3164":
            pass
        elif cbt[x,y] not in white:
            if (cbt[x,y] == "\u265C") or (cbt[x,y] == "\u265B"):
                checkcount+=1
                break
            else:
```

```
                break
        else:
            break
for xy in [(1,1),(-1,1),(1,-1),(-1,-1)]:
    x = movelist[0][0]
    y = movelist[0][1]
    while True:
        x += xy[0]
        y += xy[1]
        if not (0<=x<=7 and 1<=y<=8):
            break
        if cbt[x,y] == "\u3164":
            pass
        elif cbt[x,y] not in white:
            if (cbt[x,y] == "\u265D") or (cbt[x,y] == "\u265B"):
                checkcount+=1
                break
            else:
                break
        else:
            break
for xy in [(1,2),(2,1),(-1,2),(-2,1),(1,-2),(2,-1),(-1,-2),(-2,-1)]:
    x = movelist[0][0]
    y = movelist[0][1]
    while True:
        x += xy[0]
        y += xy[1]
        if not (0<=x<=7 and 1<=y<=8):
```

```
            break
        if cbt[x,y] not in white:
            if cbt[x,y] == "\u265E":
                checkcount+=1
                break
            else:
                break
        else:
            break
for xy in [(-1,1),(-1,-1)]:
    x = movelist[0][0]
    y = movelist[0][1]
    while True:
        x += xy[0]
        y += xy[1]
        if not (0<=x<=7 and 1<=y<=8):
            break
        if cbt[x,y] not in white:
            if cbt[x,y] == "\u265F":
                checkcount+=1
                break
            else:
                break
        else:
            break
escapelist = []
if len(movelist) > 1:
    for intloc in movelist[1:]:
```

```
matecount = 0
for xy in [(0,1),(1,0),(0,-1),(-1,0)]:
    x = intloc[0]
    y = intloc[1]
    while True:
        x += xy[0]
        y += xy[1]
        if not (0<=x<=7 and 1<=y<=8):
            break
        if cbt[x,y] == "\u3164":
            pass
        elif cbt[x,y] not in white:
            if (cbt[x,y] == "\u265C") or (cbt[x,y] == "\u265B"):
                matecount+=1
                break
            else:
                break
        else:
            break
for xy in [(1,1),(-1,1),(1,-1),(-1,-1)]:
    x = intloc[0]
    y = intloc[1]
    while True:
        x += xy[0]
        y += xy[1]
        if not (0<=x<=7 and 1<=y<=8):
            break
        if cbt[x,y] == "\u3164":
```

```
            pass
        elif cbt[x,y] not in white:
            if (cbt[x,y] == "\u265D") or (cbt[x,y] == "\u265B"):
                matecount+=1
                break
            else:
                break
        else:
            break
for xy in [(1,2),(2,1),(-1,2),(-2,1),(1,-2),(2,-1),(-1,-2),(-2,-1)]:
    x = intloc[0]
    y = intloc[1]
    while True:
        x += xy[0]
        y += xy[1]
        if not (0<=x<=7 and 1<=y<=8):
            break
        if cbt[x,y] not in white:
            if cbt[x,y] == "\u265E":
                matecount+=1
                break
            else:
                break
        else:
            break
for xy in [(-1,1),(-1,-1)]:
    x = intloc[0]
    y = intloc[1]
```

```
    while True:
        x += xy[0]
        y += xy[1]
        if not (0<=x<=7 and 1<=y<=8):
            break
        if cbt[x,y] not in white:
            if cbt[x,y] == "\u265F":
                matecount+=1
                break
            else:
                break
        else:
            break
for xy in [(0,1),(1,0),(0,-1),(-1,0),(1,1),(-1,1),(1,-1),(-1,-1)]:
    x = intloc[0]
    y = intloc[1]
    while True:
        x += xy[0]
        y += xy[1]
        if not (0<=x<=7 and 1<=y<=8):
            break
        if cbt[x,y] not in white:
            if cbt[x,y] == "\u265A":
                matecount+=1
                break
            else:
                break
        else:
```

```
                    break
            if matecount == 0:
                escapelist.append(location((intloc[0],intloc[1])))
if color == "black":
    for i in range(len(clt)):
        if clt[i] == "\u265A":
            king = i
            break
    kingstart = listcheck(king)
    king = location(kingstart)
    movelist = [(king[0],king[1])]
    for xy in [(1,1),(-1,1),(1,-1),(-1,-1),(0,1),(1,0),(0,-1),(-1,0)]:
        x = king[0]
        y = king[1]
        while True:
            x += xy[0]
            y += xy[1]
            if not (0<=x<=7 and 1<=y<=8):
                break
            if cbt[x,y] not in black:
                movelist.append((x,y))
                break
            else:
                break
    checkcount = 0
    for xy in [(0,1),(1,0),(0,-1),(-1,0)]:
        x = movelist[0][0]
        y = movelist[0][1]
```

```
    while True:
        x += xy[0]
        y += xy[1]
        if not (0<=x<=7 and 1<=y<=8):
            break
        if cbt[x,y] == "\u3164":
            pass
        elif cbt[x,y] not in black:
            if (cbt[x,y] == "\u2656") or (cbt[x,y] == "\u2655"):
                checkcount+=1
                break
            else:
                break
        else:
            break
for xy in [(1,1),(-1,1),(1,-1),(-1,-1)]:
    x = movelist[0][0]
    y = movelist[0][1]
    while True:
        x += xy[0]
        y += xy[1]
        if not (0<=x<=7 and 1<=y<=8):
            break
        if cbt[x,y] == "\u3164":
            pass
        elif cbt[x,y] not in black:
            if (cbt[x,y] == "\u2657") or (cbt[x,y] == "\u2655"):
                checkcount+=1
```

```
                break
            else:
                break
        else:
            break
for xy in [(1,2),(2,1),(-1,2),(-2,1),(1,-2),(2,-1),(-1,-2),(-2,-1)]:
    x = movelist[0][0]
    y = movelist[0][1]
    while True:
        x += xy[0]
        y += xy[1]
        if not (0<=x<=7 and 1<=y<=8):
            break
        if cbt[x,y] not in black:
            if cbt[x,y] == "\u2658":
                checkcount+=1
                break
            else:
                break
        else:
            break
for xy in [(1,1),(1,-1)]:
    x = movelist[0][0]
    y = movelist[0][1]
    while True:
        x += xy[0]
        y += xy[1]
        if not (0<=x<=7 and 1<=y<=8):
```

```
            break
        if cbt[x,y] not in black:
            if cbt[x,y] == "\u2659":
                checkcount+=1
                break
            else:
                break
        else:
            break
escapelist = []
if len(movelist) > 1:
    for intloc in movelist[1:]:
        matecount = 0
        for xy in [(0,1),(1,0),(0,-1),(-1,0)]:
            x = intloc[0]
            y = intloc[1]
            while True:
                x += xy[0]
                y += xy[1]
                if not (0<=x<=7 and 1<=y<=8):
                    break
                if cbt[x,y] == "\u3164":
                    pass
                elif cbt[x,y] not in black:
                    if (cbt[x,y] == "\u2656") or (cbt[x,y] == "\u2655"):
                        matecount+=1
                        break
                    else:
```

```
                break
            else:
                break
    for xy in [(1,1),(-1,1),(1,-1),(-1,-1)]:
        x = intloc[0]
        y = intloc[1]
        while True:
            x += xy[0]
            y += xy[1]
            if not (0<=x<=7 and 1<=y<=8):
                break
            if cbt[x,y] == "\u3164":
                pass
            elif cbt[x,y] not in black:
                if (cbt[x,y] == "\u2657") or (cbt[x,y] == "\u2655"):
                    matecount+=1
                    break
                else:
                    break
            else:
                break
    for xy in [(1,2),(2,1),(-1,2),(-2,1),(1,-2),(2,-1),(-1,-2),(-2,-1)]:
        x = intloc[0]
        y = intloc[1]
        while True:
            x += xy[0]
            y += xy[1]
            if not (0<=x<=7 and 1<=y<=8):
```

```
            break
        if cbt[x,y] not in black:
            if cbt[x,y] == "\u2658":
                matecount+=1
                break
            else:
                break
        else:
            break
for xy in [(1,1),(1,-1)]:
    x = intloc[0]
    y = intloc[1]
    while True:
        x += xy[0]
        y += xy[1]
        if not (0<=x<=7 and 1<=y<=8):
            break
        if cbt[x,y] not in black:
            if cbt[x,y] == "\u2659":
                matecount+=1
                break
            else:
                break
        else:
            break
for xy in [(0,1),(1,0),(0,-1),(-1,0),(1,1),(-1,1),(1,-1),(-1,-1)]:
    x = intloc[0]
    y = intloc[1]
```

```
                while True:
                    x += xy[0]
                    y += xy[1]
                    if not (0<=x<=7 and 1<=y<=8):
                        break
                    if cbt[x,y] not in black:
                        if cbt[x,y] == "\u2654":
                            matecount+=1
                            break
                        else:
                            break
                    else:
                        break
            if matecount == 0:
                escapelist.append(location((intloc[0],intloc[1])))
    if checkcount == 0:
        return kingstart, escapelist, 0
    elif (checkcount != 0) and (escapelist != []):
        return kingstart, escapelist, 1
    else:
        return kingstart, escapelist, -1


def location(moveinput):  #Returning the vector of a chess notation from a list by scanning a copy of the board replaced by chess locations instead of chess pieces for any hits
    notationboard = ["a8","b8","c8","d8","e8","f8","g8","h8",
                     "a7","b7","c7","d7","e7","f7","g7","h7",
                     "a6","b6","c6","d6","e6","f6","g6","h6",
                     "a5","b5","c5","d5","e5","f5","g5","h5",
                     "a4","b4","c4","d4","e4","f4","g4","h4",
```

```
                "a3","b3","c3","d3","e3","f3","g3","h3",
                "a2","b2","c2","d2","e2","f2","g2","h2",
                "a1","b1","c1","d1","e1","f1","g1","h1"]
    notationarray = chararray((9,9), itemsize=2, unicode=True)
    notationarray[:8,0] = ["8","7","6","5","4","3","2","1"]
    notationarray[0,1:] = notationboard[0:8]
    notationarray[1,1:] = notationboard[8:16]
    notationarray[2,1:] = notationboard[16:24]
    notationarray[3,1:] = notationboard[24:32]
    notationarray[4,1:] = notationboard[32:40]
    notationarray[5,1:] = notationboard[40:48]
    notationarray[6,1:] = notationboard[48:56]
    notationarray[7,1:] = notationboard[56:64]
    notationarray[8,:] = ["-","a"," b"," c"," d"," e"," f"," g","h "]


    if type(moveinput) is str:
        x = 0
        while x<9:
            y = 0
            while y<9:
                if notationarray[x,y] == moveinput:
                    return (x, y)  #Returns vector location if input is chess location
                y+=1
            x+=1
    if type(moveinput) is tuple:
        return notationarray[moveinput[0], moveinput[1]]   #Returns chess location if input is
vector location
    return
```

```
def listcheck(move):    #Returns index of chess notation by scanning a copy of the main list for any hits, returns chess notation if input is an index
    notationboard = ["a8","b8","c8","d8","e8","f8","g8","h8",
                     "a7","b7","c7","d7","e7","f7","g7","h7",
                     "a6","b6","c6","d6","e6","f6","g6","h6",
                     "a5","b5","c5","d5","e5","f5","g5","h5",
                     "a4","b4","c4","d4","e4","f4","g4","h4",
                     "a3","b3","c3","d3","e3","f3","g3","h3",
                     "a2","b2","c2","d2","e2","f2","g2","h2",
                     "a1","b1","c1","d1","e1","f1","g1","h1"]
    if type(move) is str:
        for x in range(len(notationboard)):
            if move == notationboard[x]:
                return x
    if type(move) is int:
        return notationboard[move]
    return


def timer(remtime, chesstime, move):  #Countdown timer that updates itself every second, with 5 seconds of grace time before countdown
    mins, secs = divmod(chesstime, 60)
    clock = "{:02d}:{:02d}".format(mins, secs)
    print(clock, end="\r")
    time.sleep(5)
    while chesstime>0 and not move.is_set():
        mins, secs = divmod(chesstime, 60)
        clock = "{:02d}:{:02d}".format(mins,secs)
        print(clock, end="\r")
        time.sleep(1)
```

```
        chesstime-=1
    mins, secs = divmod(chesstime, 60)
    clock = "{:02d}:{:02d}".format(mins, secs)
    print(clock)
    remtime.put(int(chesstime))
    return

def moveinput(result, move):  #Get move input from player, if finished it stops the timer
    moveinput = input("Input move: ").lower()
    move.set()
    result.put(moveinput)
    return

def board(clist, color="white"):  #Printing the board based on the color of the current player
    chessboard = chararray((9,9), itemsize=2, unicode=True)
    if color == "white":
        chessboard[:8,0] = ["8","7","6","5","4","3","2","1"]
        chessboard[8,:] = ["-","a"," b"," c"," d"," e"," f"," g","h "]
    if color == "black":
        clist = clist[::-1]
        chessboard[:8,0] = ["1","2","3","4","5","6","7","8"]
        chessboard[8,:] = ["-","h"," g"," f"," e"," d"," c"," b","a "]
    chessboard[0,1:] = clist[0:8]
    chessboard[1,1:] = clist[8:16]
    chessboard[2,1:] = clist[16:24]
    chessboard[3,1:] = clist[24:32]
    chessboard[4,1:] = clist[32:40]
    chessboard[5,1:] = clist[40:48]
```

```
    chessboard[6,1:] = clist[48:56]
    chessboard[7,1:] = clist[56:64]
    return chessboard


def login():  #Account login
    try:
        with open("chessconsoleaccts.dat", "r") as accts:
            pass
    except FileNotFoundError:
        with open("chessconsoleaccts.dat", "w") as accts:
            pass
    #Checking if file exists, if not then create new .dat file
    logcheck1 = 0
    while logcheck1 == 0:
        user1ls = input("\nPlayer 1 (White), login or signup? ").lower()
        if user1ls == "login":
            account1 = 0
            while account1 == 0:
                print("\nPlayer 1 (White), input account details: ")
                user1 = input("Input Username (20 chars): ")
                pass1 = gp.getpass(str("Input Password (20 chars): "))   #Hiding password input
                if (len(user1) > 20 or user1 == "") or (len(pass1) > 20 or pass1 == ""):   #Checking if
username and password input is valid
                    print("\nLogin data invalid, returning to login or signup screen.")
                    account1 = -1   #Value to return to login screen
                    break
                with open("chessconsoleaccts.dat", "r") as accts:   #Scanning file for username and
password pair
                    while True:
```

```
                userpass = accts.read(55)
                if not userpass:
                    break
                if (user1.ljust(20).strip() == userpass[:20].strip()) and (pass1.ljust(20).strip() == userpass[20:40].strip()):
                    account1 = 1   #Updating value to true if account is valid
                    break
            if account1 == 0:
                print("\nLogin data not in database, returning to login or signup screen.")
                account1 = -1    #Value to return to login screen
        if account1 == 1:
            logcheck1 = 1    #Updating value to true if login is valid
    elif user1ls == "signup":
        account1 = 0
        while account1 == 0:
            print("\nPlayer 1 (White), input new account details: ")
            user1 = input("Input New Username (20 chars): ")
            pass1 = gp.getpass(str("Input New Password (20 chars): "))  #Hiding password input
            if (len(user1) > 20 or user1 == "") or (len(pass1) > 20 or pass1 == ""):   #Checking if username and password input is valid
                print("\nData invalid, returning to login or signup screen.")
                account1 = -1   #Value to return to login screen
                break
            with open("chessconsoleaccts.dat", "r") as accts:   #Scanning file if username is not in use
                while True:
                    userpass = accts.read(40)
                    if not userpass:
                        account1 = 1   #Updating value to true if account is valid
```

```
                    break
                if (user1.ljust(20).strip() == userpass[:20].ljust(20).strip()):
                    account1 = 0
                    print("\nUsername already in use, please input again. ")
                    break
        if account1 == 1:
            with open("chessconsoleaccts.dat", "a") as accts:
                accts.write(user1.ljust(20)+pass1.ljust(20)+"0".ljust(5)+"0".ljust(5)+"0".ljust(5))
#Add username and password pair to database
            logcheck1 = 1   #Updating value to true if signup is valid
    else:
        print("\nInvalid command. Input again.")
print("")


logcheck2 = 0
while logcheck2 == 0:
    user2ls = input("\nPlayer 2 (Black), login or signup? ").lower()
    if user2ls == "login":
        account2 = 0
        while account2 == 0:
            print("\nPlayer 2 (Black), input account details: ")
            user2 = input("Input Username (20 chars): ")
            pass2 = gp.getpass(str("Input Password (20 chars): ")) #Hiding password input
            if (len(user2) > 20 or user2 == "") or (len(pass2) > 20 or pass2 == "") or
(user2.ljust(20).strip() == user1.ljust(20).strip()):   #Checking if username and password input is
valid, and if second user account is different from first user account
                print("\nLogin data invalid, returning to login or signup screen.")
                account2 = -1  #Value to return to login screen
                break
```

```
            with open("chessconsoleaccts.dat", "r") as accts:   #Scanning file for username and password pair
                while True:
                    userpass = accts.read(55)
                    if not userpass:
                        break
                    if (user2.ljust(20).strip() == userpass[:20].strip()) and (pass2.ljust(20).strip() == userpass[20:40].strip()):
                        account2 = 1   #Updating value to true if account is valid
                        break
            if account2 == 0:
                print("\nLogin data not in database, returning to login or signup screen.")
                account2 = -1   #Value to return to login screen
        if account2 == 1:
            logcheck2 = 1   #Updating value to true if login is valid
    elif user2ls == "signup":
        account2 = 0
        while account2 == 0:
            print("\nPlayer 2 (Black), input new account details: ")
            user2 = input("Input New Username (20 chars): ")
            pass2 = gp.getpass(str("Input New Password (20 chars): "))
            if (len(user2) > 20 or user2 == "") or (len(pass2) > 20 or pass2 == ""):
                print("\nData invalid, returning to login or signup screen.")
                account2 = -1   #Value to return to login screen
                break
            with open("chessconsoleaccts.dat", "r") as accts:   #Scanning file if username is not in use
                while True:
                    userpass = accts.read(40)
```

```
                if not userpass:
                    account2 = 1   #Updating value to true if account is valid
                    break
                if (user2.ljust(20).strip() == userpass[:20].ljust(20).strip()):
                    account2 = 0
                    print("\nUsername already in use, please input again. ")
                    break
        if account2 == 1:
            with open("chessconsoleaccts.dat", "a") as accts:
                accts.write(user2.ljust(20)+pass2.ljust(20)+"0".ljust(5)+"0".ljust(5)+"0".ljust(5))
#Add username and password pair to database
            logcheck2 = 1   #Updating value to true if signup is valid
        else:
            print("\nInvalid command. Input again.")
    print("")


    print(f"\nWelcome, {user1} and {user2}!\n")
    return user1, user2


def updateleaderboard(p1n, p1w, p1l, p1d, p2n, p2w, p2l, p2d):
    newdata = ""
    unsortedboard = {}
    with open("chessconsoleaccts.dat", "r") as accts:
        while True:
            data = accts.read(55)
            if not data:
                break
            if (p1n.ljust(20).strip() == data[:20].ljust(20).strip()):   #Scanning for account of player 1
and updating their win-loss-draw values
```

```
            data =
data[:20].ljust(20)+data[20:40].ljust(20)+str(int(data[40:45])+p1w).ljust(5)+str(int(data[45:50])+
p1l).ljust(5)+str(int(data[50:55])+p1d).ljust(5)

        if (p2n.ljust(20).strip() == data[:20].ljust(20).strip()):   #Scanning for account of player 2
and updating their win-loss-draw values

            data =
data[:20].ljust(20)+data[20:40].ljust(20)+str(int(data[40:45])+p2w).ljust(5)+str(int(data[45:50])+
p2l).ljust(5)+str(int(data[50:55])+p2d).ljust(5)

        unsortedboard[data[:20].ljust(20).strip()] = 1*int(data[40:45]) + 0.5*int(data[50:55])
#Calculating points: 1 per win, 0.5 per draw, 0 per loss

        newdata = newdata + data   #Copying entire dataset

    with open("chessconsoleaccts.dat", "w") as accts:

        accts.write(newdata)   #Replacing dat file with updated values

    leaderboard = dict(sorted(unsortedboard.items(), key=lambda value: -value[1]))  #Sorting data
by value of points, highest to lowest, using a dictionary

    leaderboardlist = list(leaderboard.items()) #Turning dictionary into a list

    print("\n\n\nLeaderboard: ")

    for i in range(5):   #Fetching top 5 from data

        placement = leaderboardlist[i] if i<len(leaderboardlist) else ("---", "---")   #Empty bracket
replacement if data has less than 5 entries

        print(f"\n{i+1}. {placement[0].ljust(22)}     | Point Rating: {placement[1]}")

    print("\n")

    return


print(startchess())
```

C. Work Breakdown –

Student Name	Tasks Assigned	Percentage of the Work Contribution
TY, Josh Angelo	• Initial project proposal • IPO Model • Base Design • Designing Chessboard • Designing of chess moves • Methodology • Results • Finalization • Script/content creation • Recording of Video	40%
TAN, Irabelle Pristine	• Initial project proposal • Project description • Designing account system • Designing of leaderboard	30%

	• Introduction • Discussion of results • Finalization • Creation of presentation • Recording of video	
BLAS, Harold Sebastian	• Initial project proposal • Designing time and input thread • Hierarchy chart • Flowchart • Conclusion • Recording of video • Editing of video	30%

D. Personal Data Sheet –

Researcher 1

Josh Angelo Paralejas Ty
2728 Tanguile Street, United Hills Village
Barangay San Martin de Porres, Parañaque City
Mobile: (+63)977-099-9139; Email: josh_ty@dlsu.edu.ph

Personal Information:

Date of Birth : July 2, 2004

Sex : Male

Nationality : Filipino

Education:

Makati Hope Christian School

Elementary and High School Graduate | June 2010 - June 2022

K2 - Grade 12

- Valedictorian in Elementary and Salutatorian in High School
- Top 1 in STEM Strand
- Student Excellence Awards in Mathematics, Scholastics, and the Science, Technology, Engineering, and Mathematics Program

Notable Achievements:

- Philippines Department of Science & Technology ~ Youth Excellence in Science (YES) Awards – Years 2016, 2017, 2018, 2019, & 2020 – 5 Medals Total
- 2019 World Mathematics Invitational Final Round– July 18, 2019, Japan – Gold Medal

Interests:

Hobbies: Anime, Manga, Gacha Games, Arts & Design

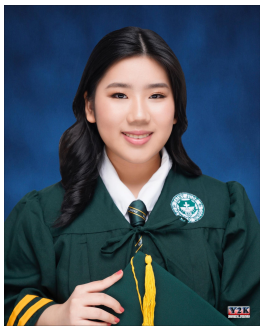
Fields: Science, Mathematics, History, Vexillology, Computer Technologies, Engineering

Technologies: Artificial Intelligence, Machine Learning, Future Technologies, Automation

Researcher 2

Irabelle Pristine Yap Tan

Mobile: +63 9233290488 Email: irabelle_tan@dlsu.edu.ph



Personal Information:

Date of Birth : October 5, 2003

Sex : Female

Nationality : Filipino

Education:

Saint Jude Catholic School | 2008-2022

Senior High School

Graduated with high honors

2020-2022

Junior High School

Graduated with honors

2016-2020

Achievements:

Outstanding Innovator Award (2022)

Outstanding Performance in a Specific Discipline - Chinese (2022)

Outstanding Service Award (2022)

Researcher 3

Harold Sebastian B. Blas
2424 P. Zamora Street,
Barangay 107, Pasay City
Mobile: (+63)915-711-0691; Email: harold_blas@dlsu.edu.ph

Personal Information:

Date of Birth : January 11, 2004

Sex : Male

Nationality : Filipino

Education:

St. Mary's Academy Pasay City

Elementary and High School Graduate | June 2012 - March 2020

Grade 3 - Grade 10

- Deportment Awardee
- Basketball Varsity Team

Colegio de San Juan de Letran

Sebior High School Graduate August 2020 - May 27

Grade 11 & Grade 12

Interests:

Hobbies: Online games, Going to the gym, Basketball, Driving, Travelling

Fields: Machine Learning, Computer Science, Biomedical Engineering, Environmental Engineering, Artificial Intelligence

E. Additional Links -

a. Github - https://github.com/Josh-Ty/pl3DionysusProject

b. Trello - https://trello.com/b/XRIgHAHj/pl3dionysusproject

c. Youtube Demo - https://youtu.be/8Vn0MEJqtYw