HERITAGE XPERIENTIAL LEARNING SCHOOL

Formerly known as The Heritage School, Gurgaon

COMPUTER SCIENCE (083)

Digital Chess

{Project Report}

(as per the guidelines of CBSE for the session 2020-21)

Particulars of Student:

Name: Rajveer Sodhi

Roll No: 21

Class: XII - E

Particulars of the Teacher

Name: HARMEET KAUR

Designation: PGT, COMPUTER SCIENCE

Contents

CERTIFICATE

This is to certify that the Python project entitled **Digital Chess**, submitted by **Rajveer Sodhi, Grade 12 (Roll No. 21)** is the bonafide work of the student (as per the guidelines of the CBSE) completed under my guidance for the session 2019-20.

NAME: Rajveer Sodhi

ROLL NO: 21

CLASS: XII - E

BOARD ROLL NO: _____

_____

Ms. Harmeet Kaur
PGT Computer Science

ACKNOWLEDGEMENTS

Regardless of the effort I put into this project from my side, a significant aspect of the completion and success of the project is due to my teachers, peers, and their unbreaking support.

It is through the following lines that I would like to express my sincere gratitude towards my fellow sojourners in the class without whom it would have been impossible to make this final project.

Thanks to all my classmates who helped me through the process and selflessly guided me whenever I got stuck. Having such amicable, symbiotic relationships with my peers has proven to be enormously helpful. Making this project would have undoubtedly been impossible without their help, and I don't think I could not have asked for better colleagues.

And needless to say, I hold the most amount of gratitude for our venerated mentor, Ms Harmeet Kaur. Thanks to her for creating a friendly and nurturing environment which effortlessly allowed us to explore new topics and hone our skills. Her constant encouragement, enthusiastic engagement, and kind words always motivated me to try to go beyond the classroom walls and push my limits so as to achieve my level best. She always provided resources and helped us out with logic whenever we were stuck.

It is with immense pleasure and gratitude that I present this project, a culmination of not just code written on my laptop, but so many other aspects and effort by those who've stood by me throughout the academic year as well.

Rajveer Sodhi

XII - E

MAIN REPORT

INTRODUCTION

Introduction:

This is a report for the joint working of two programs to run **Digital Chess**. The main program runs the game of chess. It is an in-console game playable by two players at a time. The supplementary program presents a graphical user interface. This allows the user to access records of previously played games that are stored in a top 10 leaderboard in a MySQL table.

Chess is an extremely popular game that requires high levels of algorithmic thinking and cognitive skills. The objective of the game is to eliminate the opponent's king using the 16 characteristic pieces given to each player. Every type of piece has a unique style of moving across the board. Playing this game can help sharpen a player's predictive and decision-making skills. Games can last from anywhere between 7 minutes to over 2 hours, and the design of the game keeps every minute of gameplay enthralling.

My program allows players to achieve everything the game conventionally has to offer but in a more accessible, intuitive package. The fact that this can run on any portable machine capable of executing Python code allows players to start a game in places one normally can't. Moreover, important conditions and situations in the game, such as the legal movement of pieces and whether or not the king is in check, is automatically determined by the code. This allows the users to focus more on their game itself rather than its critical yet peripheral aspects. For example, consider the chess interface. Each square on the board has a coordinate such as 1A, 2B, and so on up till 8H. The program asks you to enter the current coordinate of the piece you want to move and then the coordinate of where you want to move it to. If the latter coordinate is not legal, the program shows "Invalid move!" and asks

you to re-enter a value. On killing a player, the program prints, say, “White Pawn (4a) killed Black Pawn (5b)”, and moves the killed pawn to the side of the board. My user interface that goes along with this game also allows the players to view the rules of either my program or chess at any given moment.

Further, this program, as mentioned, comes bundled with a leaderboard system that ranks games based on the number of moves played to win. The data for this is stored in a MySQL table. This leaderboard can be accessed through the interface, from which I can either view the records in a database or edit/delete them. The leaderboard system entices users to constantly want to improve their score and keep on playing on the program. This effect can be further enhanced by potentially adding a sort of virtual reward system, where the players’ ranking on the leaderboard could account for specific redeemable rewards. Apart from this, after each game, the player can choose to save a transcript of the game. This will allow him/her to go back to that transcript to review his strategies. Users can open my user interface and access the transcript (and related statistics) about any game present in the leaderboard.

The most exciting and important part of the program, however, is ultimately the usage of the data that has been stored to train an AI model that can play chess against the user in a single-player mode. The transcript is stored in a logical format in a .txt file that can easily be converted into a file type that is readable by the computer. As the number of games played by the user increases, the amount of this data for the computer to access will also increase. The computer can recognise patterns and strategies used by players over and over and correlate them with the outcomes they achieve. The computer can also potentially create user profiles, recognising specific players by their names to relate commonly used moves to them. The collected data will help the computer build a machine learning model that understands the objective of the game and how it can be played. It would be able to make decisions regarding what move to make instantaneously after every alternating move by the user. Subsequently, after training the model, difficulty levels can also be implemented. This would allow both the play of the user as well as the computer to improve over time simultaneously.

***Data Dictionary*: Using MySQL**

The backend of this project is handled through a MySQL database named '**Chess1**'. A table '**Leaderboard**' in it holds the top 10 records for the fastest won games recorded on the program.

```
mysql> show tables in chess1;
+------------------+
| Tables_in_chess1 |
+------------------+
| leaderboard      |
+------------------+
1 row in set (0.05 sec)
```

This table has 5 columns, with a maximum of 10 records:

- **Game_Name:** The name of the chess game. The transcript of the game is also saved as a .txt file of the same name.
- **Player1:** The name of the player controlling the white pieces.
- **Player2:** The name of the player controlling the black pieces.
- **Winner:** The name of the player that won the game in question.
- **Moves:** The number of moves made by both the players in the entire game.

```
mysql> desc leaderboard;
+-----------+---------------+------+-----+---------+-------+
| Field     | Type          | Null | Key | Default | Extra |
+-----------+---------------+------+-----+---------+-------+
| Game_Name | char(15)      | YES  |     | NULL    |       |
| Player1   | char(12)      | YES  |     | NULL    |       |
| Player2   | char(12)      | YES  |     | NULL    |       |
| Winner    | char(12)      | YES  |     | NULL    |       |
| Moves     | decimal(10,0) | YES  |     | NULL    |       |
+-----------+---------------+------+-----+---------+-------+
5 rows in set (0.01 sec)
```

The records in the table Leaderboard are not in order given that MySQL tables are unordered in nature. However, they are presented in the interface of the application in order using the *'order by moves'* clause. As for how the table is maintained, if the number of moves registered in a game are less than the highest number present in the table, the record is added into it. If the table already has 10 records, the tenth one is deleted to make space

for the newest entry. On top of this, the user of my application gets the option to either delete entire records or change the name of players in a pre-existing record in the table through the app at the front end.

Skills and Modules Used:

The making of the chess game required writing complex algorithms with elementary sets of code. The simplification of certain manual aspects of the code using conditional statements and loops so as to make the code more efficient required a lot of creative problem solving, foresight, and a deep understanding of Python logic and modules.

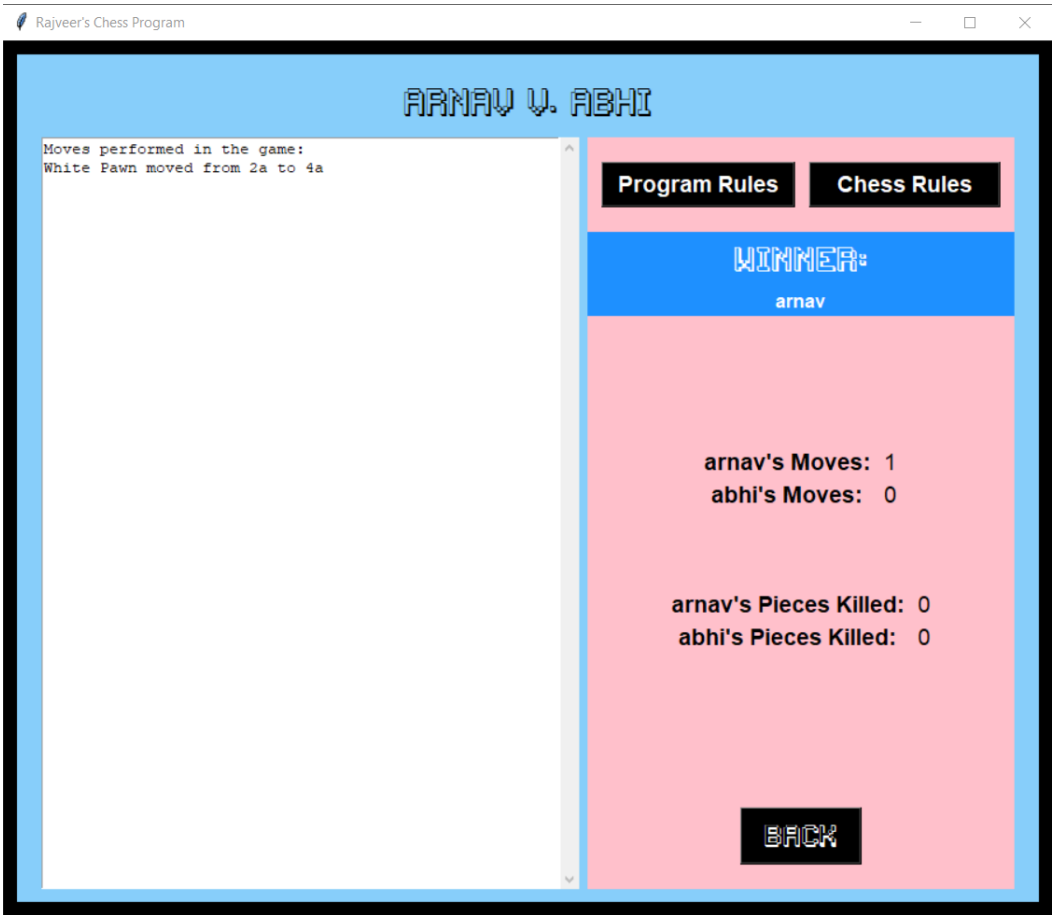
This program required strong knowledge of :

- Python fundamentals,
- MySQL fundamentals,
- the logic and tracking of functions and looping constructs,
- Tkinter fundamentals,
- Interface Design fundamentals, and
- Global variables - which were used extensively in the project.

Modules used to accomplish all aspects of our project include:

- **Tkinter**

  It is Python's de-facto standard GUI (Graphical User Interface) package. The output of my entire code to manage the chess game's leaderboard is being displayed through Tkinter windows. There are a total of 6 windows, each having a different function. The quality, pleasing design, layout, and placements of the 130+ widgets in this app would only be possible through the extensive usage of this module. The most dense use of the module is reflected in the window shown below, which boasts a total of 59 widget placements including the sub-pages present, in addition to the several configurations made to define the fonts, text sizes, and background colours through the syntax of this module.



❖ **PyMySQL**

This is an incredibly useful module used to connect MySQL servers to Python programs. This connection allows the user to access and retrieve content from SQL databases. It further allows for direct control of data stored in the database, such as making modifications to table records. The development of the leaderboard - and hence the entire graphical application - would have been impossible without this module. It has been used immensely to not only store records of the chess games in the leaderboard along with vital information about them, but also allow the user of this app to make modifications to the table in question. A clear example of this is shown in the screenshot below, where a user has the ability to select a game present in the leaderboard and alter the names of the players in the same.

App Logic Flow: Leaderboard Management GUI

The application has six windows in total:

- **Landing Page**

  The landing page reads the title of the application, '**CHESS**'. Under which, there are three buttons that allow the user to either quit the app, or navigate to the other pages. These pages are named '**Leaderboard**' and '**View Games**'. On pressing the first, the widgets on the screen rearrange to ask the user to either edit the leaderboard or view it. Whereas, the latter asks the user to choose one of the game records present in the leaderboard to view its details.

- **View Leaderboard Page**

  On opening this page, the user is presented with the leaderboard as it stands in the MySQL database in a tabular format. This is done by retrieving the records from MySQL table '**Leaderboard**' and inserting its columns' values into disabled Entry boxes which are then placed in a tabular format through looping constructs.

- **Edit Leaderboard Page**

  This page presents the user with the option to choose one of the leaderboard's records. This is achieved by extracting a tuple of the '**Game_Name**' column in the

MySQL table and putting its values into radio buttons. After that, they may press any of the two buttons below - corresponding to respective MySQL commands on the table '**Leaderboard**' - to either delete it from the MySQL table or change the name of the players in the game.

- **View Game Page**

  The View Game page shows details of the game selected from the Top 10 Leaderboard. The page is divided into two distinct columns. The column on the left is a vertically-scrollable text field that holds the transcript (stored on the computer as a .txt file) of every single move made in the game in question. The right column, right off the bat, offers the user to view either the general rules for chess or the rules they must know to be able to run my program. Below the buttons that offer to show these rules, is the name of the winner of the game, as extracted from the corresponding MySQL table record. Below that are the number of moves made by each team and the number of pieces killed from that team. These numbers are calculated under a simple for-loop that reads the lines of the transcript.

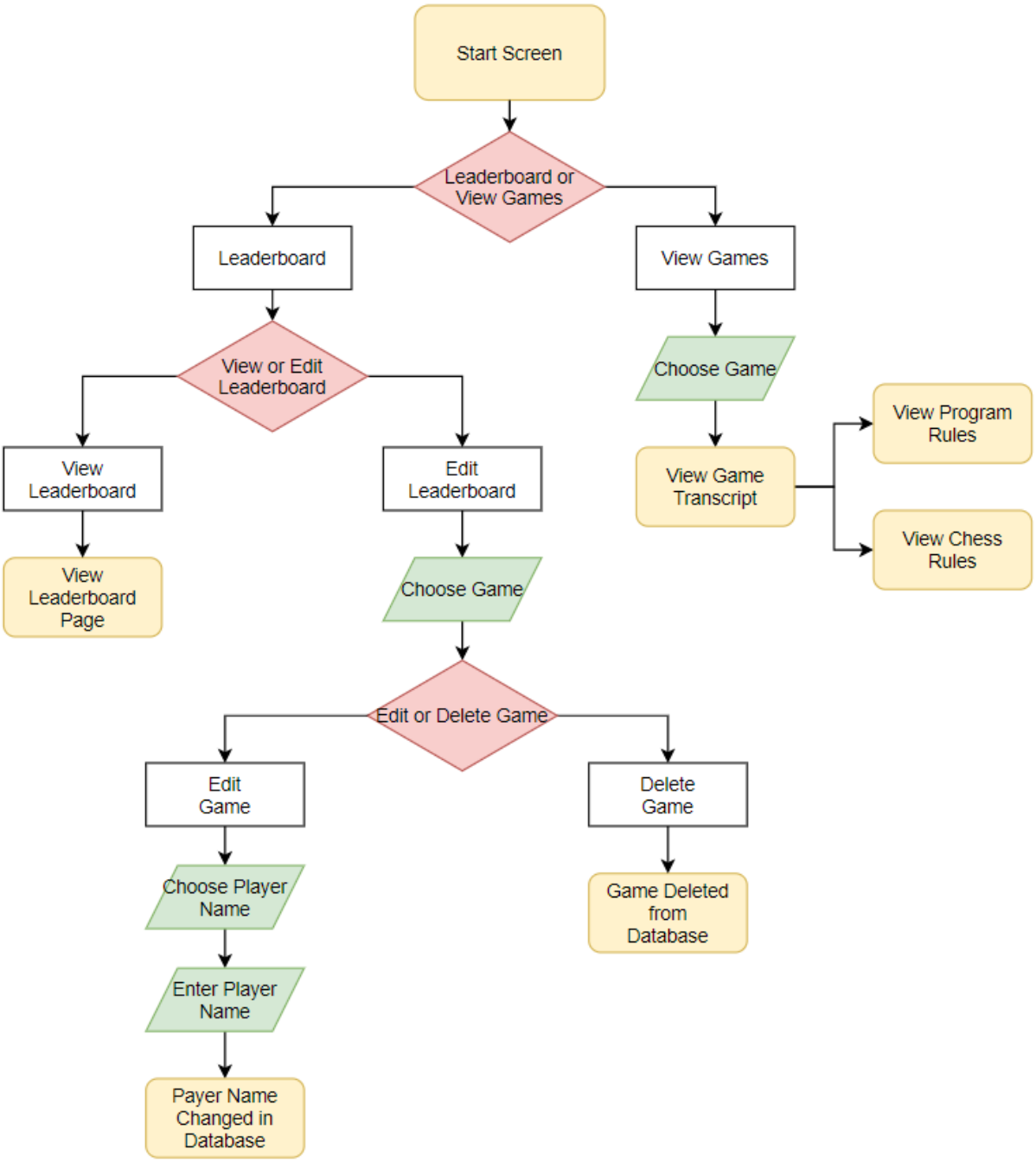
- **Program Rules Page**

  This is a simple, text-based page reading the few rules regarding the working of my specific chess program that the user may use in their gameplay, such as how to declare or call a draw in the game.

- **Chess Rules Page**

  Like the Program Rules Page, this is also a simple, text-based page listing information and rules about the game of chess in general, such as the movement of all the pieces and the ultimate motive of the game.

- **Flow Chart**



App Logic Flow: Digital Chess

- **Chess Board Logic**

  This project displays an 8*8 array with the white and black chess pieces at the bottom and top edge of the board respectively. The players can alternately move their pieces, starting with the white team. In order to do so, the grid is made using an umbrella list with 8 sublists in it, each with 8 items. The items in these lists

represent each box of the chess board and are printed in the necessary format. This format makes it easier to move the shown pieces up/down or left/right by adding or subtracting a value to the row or column values hence generated using this method.

- **Piece Movement Logic**

To move a piece, the player must enter the board coordinates of the piece they want to move, and subsequently those of where they want to move the piece to. To accomplish this, the program asks for the initial coordinates of the piece, which, once entered in the form of (row+column) - such as “2a”, “3b” - it is broken down and converted into the coordinates of that specific list item. It then reads what is the value of that list item. If the value is a blank space, or a piece belonging to the opponent, or if the piece has no valid movements in its current position, the program asks for input again, doing so until the player enters a valid coordinate.

According to what piece it is, the program is then directed to a function created for that piece (there are a total of 6 such functions created, one for each type of piece), where the user is asked to input the desired new coordinates of the piece in a similar fashion. Conditions in the function then decide whether or not the coordinates entered are valid for the piece. They also check if the piece’s initial and new coordinates, if acceptable, have any obstruction between them. Hence, similar to the initial coordinate input, the user is asked for input until a valid coordinate is entered.

- **Piece Killing Logic**

When a piece is killed, the killed piece gets displayed to the right of the board in one of the two 4*4 grids for black and white pieces respectively. This helps keep track of the game. The program also displays a message showing the killer and killing piece along with their respective coordinates. As per the rules of chess, under certain conditions, there is also the option to kill a pawn using another pawn by means of a method known as “en passant”.

As and when a player's king is put under check, a rather intricate function defined to keep track of that is enabled that disallows the player to make any move that does not immediately put the king out of check. If the player is unable to get out of that situation, they must declare defeat. The game can be stopped if any of the players enters "declare" or "draw" where they're asked to enter initial coordinates. Both of these are followed by a confirmatory input, after which the game comes to an end.

- **Additional Logic**

If a pawn reaches the opposite end of the board, it can be promoted to a higher class of pieces, such as a queen or a knight. It cannot, however, be promoted to a king. Hence, it is possible to have up to 9 queens, or 10 bishops/knights/rooks.

The user may also castle if he wishes to, that is, move the king 2 spaces towards a rook, and move the rook to the space that the king skipped over. This is the only time in the game when more than one piece moves at a time.

The working of the function that keeps track of whether or not the king of a player is under check is that it sees if the king is in the path of any of its opponents potential moves after every move made - irrespective of the team performing the move. Once the check is declared, the player can enter the move he wants to make. The program makes that move for them, and checks whether or not their king is still in check after that move. If it is, the move is reversed and the player is asked to try again.

- **File Handling Logic**

A .txt file is opened at the start of the game, to which a line is appended after every move made stating the movement of the piece. A line is similarly added every time a pawn is promoted or a piece is killed. Once the game ends, the user is asked if they want to save a transcript of the game. If they select yes, the contents of the temporary file opened at the start of the game is transferred to another file named by the user. After this, the number of lines of the file are counted to determine

whether or not the count is less than the largest count present in the MySQL table '**Leaderboard**'. If so, the program asks the user for the names of both the players playing the game and adds their record to the table. This information can be viewed by the user by running my GUI program.

Future Capabilities:

The code of this program allows for very exciting potential for future functionality of the application.

- **Integration of both Programs**

  As of now, both the programs have to be separately run to be accessed. However, future modifications to the codes can integrate them both into one program, allowing both the GUI as well as the Chess game to be run simultaneously for a better user experience.

- **Single-Player Mode**

  At its current stage, the code allows the user to store a complete transcript of all the moves performed in games of chess played. This is stored in a .txt file. Each line of the file represents a move giving the piece names, their initial positions, and their final positions. For example, *'White Pawn (4A) killed Black Queen (5B)'***.** Moreover, a MySQL table maintains a leaderboard containing the names of players and the winner. The storage of this information means that the program holds the potential to be subject to machine learning (ML) algorithms that study the moves made throughout a game, in several recorded games, to determine what strategies and situations lead to chess games being won. An artificial intelligence (AI) model can subsequently be trained that can play the game of chess with a user. This would allow for a single-player mode wherein levels of difficulty can be set. Furthermore, the more games that are played, the better this AI model will become.

- **Conversion into an App**

  The single-program as mentioned in the above point can be further improved to run as one graphic-based program only. I.e., the chess game itself can also be made graphical in nature. On top of that, the environment around the game can be developed. The whole ecosystem can be converted into an app that can be downloaded from app stores of various operating systems. Advertisements and free sampling of the gameplay can help increase user interaction that will help in training the AI model for the single-player mode. The leaderboard can hence be made universal and not just bound to games played on the system itself. A reward system can be implemented wherein the higher a user ranks on the leaderboard, the more points he/she is awarded. These points could then be used maybe as coupons or shoutouts on their social media accounts, etc.

HARDWARE AND SOFTWARE REQUIREMENTS

→ Computer System Used:

- Macbook (Retina, 12-inch, Early 2016)

  macOS Mojave, Version 10.14.13

  - **Processor:** 1.3 GHz Intel Core m7
  - **Memory:** 8 GB 1867 MHz LPDDR3

- Dell Inspiron 7560

  Windows 10 Home Single Language

  - **Processor:** Intel(R) Core(™) i7-7500U CPU @ 2.70GHz 2.90GHz
  - **Memory:** 8 GB LPDDR3

→ Software Used:

- Spyder 3.8.3
- Visual Studio Code 1.50.1
- MySQL 8.0
- Python 3.8.6

OUTPUTS AND CODING

PROGRAM:

CHESS GAME

```
''' Rajveer's Chess Program '''

#
============================================================
=================
# miscellaneous functions
#
============================================================
=================


with open ("moves.txt", "w") as c1:
    c1.write("Moves performed in the game: \n")

board = [[   "null", "null",    "null",    "null",
"null",    "null",    "null",   "null",    "null"],
# the board layout
            ["null", chr(9820), chr(9822), chr(9821),
chr(9819), chr(9818), chr(9821), chr(9822), chr(9820),
"null", " ", " ", " ", " "],
            ["null", chr(9823), chr(9823), chr(9823),
chr(9823), chr(9823), chr(9823), chr(9823), chr(9823),
"null", " ", " ", " ", " "],
            ["null", " ",       " ",       " ",       " ",
" ",       " ",       " ",       " ",       "null", " ", "
", " ", " "],
            ["null", " ",       " ",       " ",       " ",
" ",       " ",       " ",       " ",       "null", " ", "
", " ", " "],
            ["null", " ",       " ",       " ",       " ",
" ",       " ",       " ",       " ",       "null", " ", "
", " ", " "],
            ["null", " ",       " ",       " ",       " ",
" ",       " ",       " ",       " ",       "null", " ", "
", " ", " "],
            ["null", chr(9817), chr(9817), chr(9817),
chr(9817), chr(9817), chr(9817), chr(9817), chr(9817),
"null", " ", " ", " ", " "],
            ["null", chr(9813), chr(9816), chr(9813),
chr(9813), chr(9812), chr(9815), chr(9816), chr(9814),
"null", " ", " ", " ", " "],
            ["null", "null",    "null",    "null",
"null",    "null",    "null",    "null",    "null"]]
def print_board(x):
    print("\n", "       A   B   C   D   E   F   G   H",
"\n", "\n", "     +-------------------------------+", "\n",
" 8  ", "|", x[1][1], "|", x[1][2], "|", x[1][3], "|",
x[1][4], "|", x[1][5], "|", x[1][6], "|", x[1][7], "|",
x[1][8], "|", "  8       ", x[1][10], "  ", x[1][11], "  ",
```

```
x[1][12], "  ", x[1][13], "\n", "
|-------------------------------|")
    print("  7  ", "|", x[2][1], "|", x[2][2], "|", x[2][3],
"|", x[2][4], "|", x[2][5], "|", x[2][6], "|", x[2][7], "|",
x[2][8], "|", "  7       ", x[2][10], "  ", x[2][11], "  ",
x[2][12], "  ", x[2][13], "\n", "
|-------------------------------|")
    print("  6  ", "|", x[3][1], "|", x[3][2], "|", x[3][3],
"|", x[3][4], "|", x[3][5], "|", x[3][6], "|", x[3][7], "|",
x[3][8], "|", "  6       ", x[3][10], "  ", x[3][11], "  ",
x[3][12], "  ", x[3][13], "\n", "
|-------------------------------|")
    print("  5  ", "|", x[4][1], "|", x[4][2], "|", x[4][3],
"|", x[4][4], "|", x[4][5], "|", x[4][6], "|", x[4][7], "|",
x[4][8], "|", "  5       ", x[4][10], "  ", x[4][11], "  ",
x[4][12], "  ", x[4][13], "\n", "
|-------------------------------|")
    print("  4  ", "|", x[5][1], "|", x[5][2], "|", x[5][3],
"|", x[5][4], "|", x[5][5], "|", x[5][6], "|", x[5][7], "|",
x[5][8], "|", "  4       ", x[5][10], "  ", x[5][11], "  ",
x[5][12], "  ", x[5][13], "\n", "
|-------------------------------|")
    print("  3  ", "|", x[6][1], "|", x[6][2], "|", x[6][3],
"|", x[6][4], "|", x[6][5], "|", x[6][6], "|", x[6][7], "|",
x[6][8], "|", "  3       ", x[6][10], "  ", x[6][11], "  ",
x[6][12], "  ", x[6][13], "\n", "
|-------------------------------|")
    print("  2  ", "|", x[7][1], "|", x[7][2], "|", x[7][3],
"|", x[7][4], "|", x[7][5], "|", x[7][6], "|", x[7][7], "|",
x[7][8], "|", "  2       ", x[7][10], "  ", x[7][11], "  ",
x[7][12], "  ", x[7][13], "\n", "
|-------------------------------|")
    print("  1  ", "|", x[8][1], "|", x[8][2], "|", x[8][3],
"|", x[8][4], "|", x[8][5], "|", x[8][6], "|", x[8][7], "|",
x[8][8], "|", "  1       ", x[8][10], "  ", x[8][11], "  ",
x[8][12], "  ", x[8][13], "\n", "
+-------------------------------+", "\n", "\n", "       A
B   C   D   E   F   G   H", "\n")

def select(board, piece_list, y_dict, trapped, pawn, rook,
knight, bishop, queen, king, select):        # initial
coordinates selection
    global selection, breaker, init_u_coords, init_u_x,
init_u_y  , init_c_x, init_c_y, winner
    while selection not in piece_list:
        if breaker:
            break
        while trapped == True:
            try:
                init_u_coords = input("Enter the coordinates
of the piece you want to move: ")
                if init_u_coords.lower() == "declare":
# declaration of loss
                    declare_sure = input("Are you sure?
(y/n): ")
                    if declare_sure.lower() == "y":
                        print(you, "declared defeat.",
opponent, "wins!")
                        breaker, winner = True, opponent
                        break
                    else:
```

```
                        print(you, "withdrew their
declaration.")
                        select(board, piece_list, y_dict,
trapped, pawn, rook, knight, bishop, queen, king, select)
                        break
                elif init_u_coords.lower() == "draw":
# calls to end the match in a draw
                    print(you, "is calling for a draw!
Does", opponent, "agree? (Type draw if yes):")
                    draw_input = input()
                    if draw_input.lower() == "draw":
                        print("The game ended in a draw!")
                        breaker, winner = True, "draw"
                        break
                    else:
                        print(opponent, "Doesn't agree with
you. The match coninues.", "\n")
                        select(board, piece_list, y_dict,
trapped, pawn, rook, knight, bishop, queen, king, select)
                        break
                init_u_x, init_u_y = int(init_u_coords[0]),
init_u_coords[1].title()
                if init_u_x not in range(1,9):
                    continue
                init_c_x, init_c_y = 9-init_u_x,
y_dict[init_u_y]
                selection = board[init_c_x][init_c_y]
                if selection not in piece_list:
# checks if the coordinates entered have a piece from your
team or from the other team/ is a blank space
                    print("Error! Please choose a piece from
your team.")
                    continue
                trapped, trap_breaker = False, False
# checks if the piece is surrounded, i.e., if it has any
legal moves at all. if it doesn't the user won't be allowed
to choose it at all
                if selection == chr(9817):
# white pawn
                    for i in range(-1, 2, 2):
                        if init_c_x == 4 and
board[init_c_x][init_c_y+i] == chr(9823) and
board[init_c_x-1][init_c_y+i] == " ":
                            trapped, trap_breaker = False,
True
                            break
                    if trap_breaker:
                        break
                    if board[init_c_x-1][init_c_y] != " "
and board[init_c_x-1][init_c_y+1] not in black_list and
board[init_c_x-1][init_c_y-1] not in black_list:
                        trapped = True
                elif selection == chr(9823):
# black pawn
                    for i in range(-1,2,2):
                        if init_c_x == 5 and
board[init_c_x][init_c_y+i] == chr(9817) and
board[init_c_x+1][init_c_y+i] == " ":
                            trapped, trap_breaker = False,
True
                            break
```

```
                    if trap_breaker:
                        break
                    if board [init_c_x+1][init_c_y] != " "
and board[init_c_x+1][init_c_y+1] not in white_list and
board[init_c_x+1][init_c_y-1] not in white_list:
                        trapped = True
                elif selection == chr(9814) or selection ==
chr(9820):        # rook
                    if ((board[init_c_x+1][init_c_y] in
piece_list or board[init_c_x+1][init_c_y] == "null") and
(board[init_c_x-1][init_c_y] in piece_list or
board[init_c_x-1][init_c_y] == "null")) and
((board[init_c_x][init_c_y+1] in piece_list or
board[init_c_x][init_c_y+1] == "null") and
(board[init_c_x][init_c_y-1] in piece_list or
board[init_c_x][init_c_y-1] == "null")):
                        trapped = True
                elif selection == chr(9815) or selection ==
chr(9821):        # bishop
                    if (board[init_c_x+1][init_c_y+1] in
piece_list or board[init_c_x+1][init_c_y+1] == "null") and
(board[init_c_x+1][init_c_y-1] in piece_list or
board[init_c_x+1][init_c_y-1] == "null") and
(board[init_c_x-1][init_c_y-1] in piece_list or
board[init_c_x-1][init_c_y-1] == "null") and
(board[init_c_x-1][init_c_y+1] in piece_list or
board[init_c_x-1][init_c_y+1] == "null"):
                        trapped = True
                elif selection == chr(9812) or selection ==
chr(9818) or selection == chr(9819) or selection ==
chr(9813):      # king, queen
                    if ((board[init_c_x+1][init_c_y] in
piece_list or board[init_c_x+1][init_c_y] == "null") and
(board[init_c_x-1][init_c_y] in piece_list or
board[init_c_x-1][init_c_y] == "null")) and
((board[init_c_x][init_c_y+1] in piece_list or
board[init_c_x][init_c_y+1] == "null") and
(board[init_c_x][init_c_y-1] in piece_list or
board[init_c_x][init_c_y-1] == "null")) and
(board[init_c_x+1][init_c_y+1] in piece_list or
board[init_c_x+1][init_c_y+1] == "null") and
(board[init_c_x+1][init_c_y-1] in piece_list or
board[init_c_x+1][init_c_y-1] == "null") and
(board[init_c_x-1][init_c_y-1] in piece_list or
board[init_c_x-1][init_c_y-1] == "null") and
(board[init_c_x-1][init_c_y+1] in piece_list or
board[init_c_x-1][init_c_y+1] == "null"):
                        trapped = True

                if trapped == True:
                    print("Error! Please choose a piece that
can move.")
            except Exception:
                continue

def en_passant(board, new_c_x, new_c_y, print_board,
kill_dict, selection, init_u_coords ,new_u_coords, counter,
u_y_dict, white_list, black_list):     # en passant movement
for pawns
    global white_kill, black_kill, inloop, printer, passant
    while inloop:
```

```
        if counter % 2 == 0:
            kill_list, kill_add, initial_x, a,
killed_u_coord, killed_chr, opp_list, king, add =
white_kill, 5, 4, -1, "5", chr(9823), black_list, chr(9812),
black_add      # variables to make the same function
applicable to both pawns
        elif counter % 2 == 1:
            kill_list, kill_add, initial_x, a,
killed_u_coord, killed_chr, opp_list, king, add =
black_kill, 1, 5, 1, "4", chr(9817), white_list, chr(9818),
white_add
        for i in range (-1, 2, 2):
            if init_c_x == initial_x and
board[init_c_x][init_c_y+i] == killed_chr:
                if new_c_x == init_c_x+a and new_c_y ==
init_c_y+i:
                    killed_piece, killer_piece, killer,
killed = kill_dict[board[init_c_x][new_c_y]],
kill_dict[selection], board[init_c_x][init_c_y],
board[new_c_x][new_c_y]
                    killed_piece_u_coords = killed_u_coord +
u_y_dict[new_c_y]
                    board[new_c_x][new_c_y],
board[init_c_x][init_c_y+i], board[init_c_x][init_c_y] =
killer, " ", " "
                    checker(board, opp_list, king, add)
                    if selection in white_list:
                        check = white_check
                    elif selection in black_list:
                        check = black_check
                    if check:
                        print("King is under check! Invalid
move!")
                        board[new_c_x][new_c_y],
board[init_c_x][init_c_y], board[new_c_x][new_c_y+i],
inloop, counter, printer = " ", killer, killed, False,
counter - 1, False
                        break
                    else:
                        board[new_c_x][new_c_y],
board[init_c_x][init_c_y+i], board[init_c_x][init_c_y] =
killer, " ", " "
                        board[(kill_list // 4) +
kill_add][(kill_list % 4)+10] = board[init_c_x][new_c_y]
                        kill_list, inloop = kill_list + 1,
inloop
                        print("\n", killer_piece," (",
init_u_coords[0:2], ") ", "killed ", killed_piece, " (",
killed_piece_u_coords, ")", "\n", sep = "")
                        if counter % 2 == 0:
# shows the killed piece to the right of the board in a 4x4
grid for each team
                            board[(white_kill //
4)+5][(white_kill % 4)+10], white_kill =
board[new_c_x][new_c_y], white_kill + 1
                        elif counter % 2 == 1:
                            board[((black_kill) //
4)+1][(black_kill % 4)+10], black_kill =
board[new_c_x][new_c_y], black_kill + 1
                        print_board(board)
                        break
```

```
    passant = False

def new_coords_choice(y_dict):      # asks for entry of new
coordinates
    global new_c_x, new_c_y, new_u_coords, back, counter,
printer, breaker
    while True:
        try:
            new_u_coords = input("Enter the coordinates of
where you want to move the piece: ")
            if new_u_coords.lower() == "back":
                back, counter, printer = True, counter - 1,
False
                break
            else:
                new_u_x, new_u_y = int(new_u_coords[0]),
new_u_coords[1].title()
            if new_u_x not in range(1,9):           # makes
sure the row entered is between 1 and 8
                continue
            new_c_x, new_c_y = 9-new_u_x, y_dict[new_u_y]
            break
        except Exception:
            pass

def counter_checker(counter, white_list, black_list):
# defines whose turn it is and assigns the list of the
opponent's pieces accordingly
    global piece_list
    if counter % 2 == 0:
        piece_list = black_list
    if counter % 2 == 1:
        piece_list = white_list

def checker(board, white_list, black_king, white_add):
# checks whether a team's king is under check
    global black_check, white_check
    king, check = black_king, False
    for king_x in range (len(board)):
        if type(board[king_x]) == type(board):
            try:
                king_y = board[king_x].index(black_king)
                break
            except ValueError:
                pass

    for i in range(-1,2,2):             # pawn
        try:
            if board[king_x+white_add][king_y+i] ==
white_list[5]:
                check = True
                break
        except IndexError:
            pass
    for i in range (king_x+1, 9):       #white rook, queen:
straight
        if board[i][king_y] == white_list[0] or
board[i][king_y] == white_list[3]:
            check = True
        elif board[i][king_y] not in [" " , white_list[3],
white_list[0]]:
```

```
            break
    for i in range(king_x-1, 0, -1):
        if board[i][king_y] == white_list[0] or
board[i][king_y] == white_list[3]:
            check = True
        elif board[i][king_y] not in [" " , white_list[3],
white_list[0]]:
            break
    for i in range (king_y+1, 9):
        if board[king_x][i] == white_list[0] or
board[king_x][i] == white_list[3]:
            check = True
        elif board[king_x][i] not in [" " , white_list[3],
white_list[0]]:
            break
    for i in range(king_y-1, 0, -1):
        if board[king_x][i] == white_list[0] or
board[king_x][i] == white_list[3]:
            check = True

        elif board[king_x][i] not in [" " , white_list[3],
white_list[0]]:
            break
    y=king_y+1                  # white bishop, queen:
diagonal
    for x in range(king_x+1, 9):
        try:
            if board[x][y] == white_list[3] or board[x][y]
== white_list[2]:
                check = True
            elif board[x][y] not in [" ", white_list[3],
white_list[2]]:
                break
            if y < 8:
                y += 1
        except IndexError:
            break
    y=king_y+1
    for x in range(king_x-1, 0, -1):
        try:
            if board[x][y] == white_list[3] or board[x][y]
== white_list[2]:
                check = True
            elif board[x][y] not in [" ", white_list[3],
white_list[2]]:
                break
            if y < 8:
                y += 1
        except IndexError:
            break
    y=king_y-1
    for x in range(king_x-1, 0, -1):
        try:
            if board[x][y] == white_list[3] or board[x][y]
== white_list[2]:
                check = True
            elif board[x][y] not in [" ", white_list[3],
white_list[2]]:
                break
            if y > 0:
                y -= 1
```

```
            except IndexError:
                break
        y=king_y-1
        for x in range(king_x+1, 9):
            try:
                if board[x][y] == white_list[3] or board[x][y]
== white_list[2]:
                    check = True
                elif board[x][y] not in [" ", white_list[3],
white_list[2]]:
                    break
                if y > 0:
                    y -= 1
            except IndexError:
                break
        for i in range (-1,2,2):            # knight
            for m in range (-2,3,4):
                try:
                    if board[king_x+m][king_y+i] ==
white_list[1] or board[king_x+i][king_y+m] == white_list[1]:
                        check = True
                except IndexError:
                    pass
        for i in range(-1,2,2):             # king
            try:
                if board[king_x][king_y+i] == white_list[4] or
board[king_x+i][king_y] == white_list[4] or
board[king_x+i][king_y+i] == white_list[4] or
board[king_x+i][king_y-i] == white_list[4]:
                    check = True
            except IndexError:
                pass

        if king == chr(9818):
            black_check = check
        elif king == chr(9812):
            white_check = check

    def movement(kill, board, checker, new_c_x, init_c_x,
new_c_y, init_c_y, opp_list, king, add, white_list,
black_list, check):       # master function for movement of
all pieces, includes functionality for checking check as
well as the killing of pieces. will be used for movement in
90% places.
        global inloop, counter, white_check, black_check,
printer
        new_piece = board[new_c_x][new_c_y]
        board[new_c_x][new_c_y] = board[init_c_x][init_c_y]
        board[init_c_x][init_c_y] = " "
        checker(board, opp_list, king, add)
        if selection in white_list:
            check = white_check
        elif selection in black_list:
            check = black_check
        if check:
            print("King is under check! Invalid move!")
            board[init_c_x][init_c_y] = board[new_c_x][new_c_y]
            board[new_c_x][new_c_y] = new_piece
            inloop, counter, printer = False, counter - 1, False
        else:
            board[init_c_x][init_c_y] = board[new_c_x][new_c_y]
```

```
        board[new_c_x][new_c_y] = new_piece
        kill(board, new_c_x, new_c_y, print_board,
kill_dict, selection, init_u_coords ,new_u_coords, counter,
init_c_x, init_c_y)
        inloop = False
def kill(board, new_c_x, new_c_y, print_board, kill_dict,
selection, init_u_coords ,new_u_coords, counter, init_c_x,
init_c_y):              # updates board with new positions,
but if a player is killed, shows kill message and displays
the killed piece to the side of the board
    global white_kill, black_kill, king_breaker, moves_kill,
winner
    king_checker = ""
    if board[new_c_x][new_c_y] != " ":
        killed_piece, killer_piece =
kill_dict[board[new_c_x][new_c_y]], kill_dict[selection]
        if counter % 2 == 0:                '[;p'''
# shows the killed piece to the right of the board in a 4x4
grid for each team
            board[(white_kill // 4)+5][(white_kill % 4)+10],
white_kill = board[new_c_x][new_c_y], white_kill + 1
        elif counter % 2 == 1:
            board[((black_kill) // 4)+1][(black_kill %
4)+10], black_kill = board[new_c_x][new_c_y], black_kill + 1
        kill_statement = killer_piece + " (" +
init_u_coords[0:2] + ") killed " + killed_piece + " (" +
new_u_coords[0:2] + ") \n"
        print("\n", kill_statement)
        with open("moves.txt", "a") as c1:
            c1.write(kill_statement)
        moves_kill = True
        king_checker = killed_piece
    board[new_c_x][new_c_y] = board[init_c_x][init_c_y]
    board[init_c_x][init_c_y] = " "
    print_board(board)
    if king_checker == "Black King":                    # if
the killed piece is a king, the game ends
        print("The Black King has been killed! Player 1
wins!")
        with open("moves.txt", "a") as c1:
            c1.write("The Black King has been killed! Player
1 wins!")
        king_breaker, moves_kill, winner = True, True,
"Player 1"
    elif king_checker == "White King":
        print("The White King has been killed! Player 2
wins!")
        with open("moves.txt", "a") as c1:
            c1.write("The White King has been killed! Player
2 wins!")
        king_breaker, moves_kill, winner = True, True,
"Player 2"

def update_board(board, new_c_x, new_c_y, print_board,
kill_dict, selection, init_u_coords ,new_u_coords, counter,
init_c_x, init_c_y) :            # updating board with new
positions, primarily used in pawn movement
    board[new_c_x][new_c_y], board[init_c_x][init_c_y] =
board[init_c_x][init_c_y], " "
    print_board(board)
```

```
def diagonal_block(board, new_c_x, init_c_x, new_c_y,
init_c_y, change):            # blockage checker for bishop
and diagonal movement of queen
    global add, blockage
    blockage, add = False, 1
    if new_c_x == init_c_x+change:                  # checks
if its path is obstructed
        if new_c_y > init_c_y:          # down, right
            for c in range(init_c_x+1, new_c_x):
                if board[init_c_x+add][init_c_y+add] != " ":
                    blockage = True
                add += 1
        elif new_c_y < init_c_y:        # down, left
            for c in range(init_c_x+1, new_c_x):
                if board[init_c_x+add][init_c_y-add] != " ":
                    blockage = True
                add += 1
    elif new_c_x == init_c_x-change:
        if new_c_y > init_c_y:          # up, right
            for c in range(new_c_x+1, init_c_x):
                if board[init_c_x-add][init_c_y+add] != " ":
                    blockage = True
                add += 1
        elif new_c_y < init_c_y:        # up, left
            for c in range(new_c_x+1, init_c_x):
                if board[init_c_x-add][init_c_y-add] != " ":
                    blockage = True
                add += 1
def straight_block(board, new_c_x, init_c_x, new_c_y,
init_c_y):        # blockage checker for rook and straight
movement of queen
    global blockage
    blockage = False            # checks if its path is
obstructed
    if new_c_y == init_c_y:
        if init_c_x > new_c_x:
            for block in range (new_c_x+1, init_c_x):
                if board[block][new_c_y] != " ":
                    blockage = True
        elif init_c_x < new_c_x:
            for block in range (init_c_x+1, new_c_x):
                if board[block][new_c_y] != " ":
                    blockage = True
    elif new_c_x == init_c_x:
        if init_c_y > new_c_y:
            for block in range (new_c_y+1, init_c_y):
                if board[new_c_x][block] != " ":
                    blockage = True
        elif init_c_y < new_c_y:
            for block in range (init_c_y+1, new_c_y):
                if board[new_c_x][block] != " ":
                    blockage = True


# various lists, tuples, dictionaries and variables defined
for reference throughout the code
kill_dict = {chr(9820) : "Black Rook", chr(9822) : "Black
Knight", chr(9821) : "Black Bishop", chr(9819) : "Black
Queen" , chr(9818) : "Black King", chr(9823) : "Black Pawn",
chr(9814) : "White Rook", chr(9816) : "White Knight",
```

```
chr(9815) : "White Bishop", chr(9813) : "White Queen",
chr(9812) : "White King", chr(9817) : "White Pawn"}
y_dict, u_y_dict = {"A" : 1, "B" : 2, "C" : 3, "D" : 4, "E"
: 5, "F" : 6, "G" : 7, "H" : 8}, {1 : "a", 2 : "b", 3 : "c",
4 : "d", 5 : "e", 6 : "f", 7 : "g", 8 : "h"}
white_promotion_dict, black_promotion_dict = {"Rook" :
chr(9814), "Knight" : chr(9816), "Bishop" : chr(9815),
"Queen" : chr(9813)},  {"Rook" : chr(9820), "Knight" :
chr(9822), "Bishop" : chr(9821), "Queen" : chr(9819)}
white_list, black_list = (chr(9814), chr(9816), chr(9815),
chr(9813), chr(9812), chr(9817), "♙", "♖", "♘", "♗", "♕",
"♔"), (chr(9820), chr(9822), chr(9821), chr(9819),
chr(9818), chr(9823), "♟", "♜", "♞", "♝", "♛", "♚")
new_c_x, new_c_y, new_u_coords, counter, piece_list,
white_kill, black_kill, white_castling_checker,
black_castling_checker, king_breaker, black_check,
white_check, white_king, black_king, white_add, black_add,
back, inloop, printer, passant, blockage, add, winner = 0,
0, "0z", 0, [], 0, 0, True, True, False, False, False,
chr(9812), chr(9818), 1, -1, False, True, True, True, False,
1, ""

#
=============================================================
=================
# piece movement
#
=============================================================
=================

def pawn(board, print_board, init_c_x, init_c_y, black_list,
new_coords_choice, update_board, y_dict, white_list, kill,
en_passant, white_promotion_dict, black_promotion_dict,
selection, king_breaker, movement):
    global back, white_check, black_check, counter, inloop,
passant
    if counter % 2 == 0:                          # variables
that make the function usable for pawns from either team
        initial_x, a, piece_list, final_x, promotion_dict,
passant_x, opponent, opponent_king, opp_list, add, check,
king = 7, -1, black_list, 1, white_promotion_dict, 4,
chr(9823), chr(9818), black_list, black_add, white_check,
chr(9812)
    elif counter % 2 == 1:
        initial_x, a, piece_list, final_x, promotion_dict,
passant_x, opponent, opponent_king, opp_list, add, check,
king = 2, 1, white_list, 8, black_promotion_dict, 5,
chr(9817), chr(9812), white_list, white_add, black_check,
chr(9818)
    inloop, passant = True, True
    while inloop:
        new_coords_choice(y_dict)
        if back:
            inloop, back = False, False
            break

        new_piece = board[new_c_x][new_c_y]
        for i in range(-1, 2, 2):                 # calls for
en passant function
```

```
            if init_c_x == passant_x and
board[init_c_x][init_c_y+i] == opponent and new_c_y ==
init_c_y+i and board[new_c_x][new_c_y] == " ":
                en_passant(board, new_c_x, new_c_y,
print_board, kill_dict, selection, init_u_coords
,new_u_coords, counter, u_y_dict, white_list, black_list)


        if init_c_x == initial_x:           # checks for
movement if pawn has not been moved yet
            for i in range(-1, 2, 2):
                if board[init_c_x+a][init_c_y+i] in
piece_list:     # if the pawn is killing a piece
                    if new_c_x == init_c_x+a and new_c_y ==
init_c_y+i:
                        movement(kill, board, checker,
new_c_x, init_c_x, new_c_y, init_c_y, opp_list, king, add,
white_list, black_list, check)
                        break
                    elif new_c_x == init_c_x+a and new_c_y
== new_c_y and board[new_c_x][new_c_y] == " ":   # if it is
moving just one space
                        movement(update_board, board,
checker, new_c_x, init_c_x, new_c_y, init_c_y, opp_list,
king, add, white_list, black_list, check)
                        break
            if (new_c_x == init_c_x+(2*a) or new_c_x ==
init_c_x+a) and new_c_y == init_c_y and
board[new_c_x][new_c_y] == " ":     # if it is moving 2
spaces
                if new_c_x == init_c_x+(2*a):
                    if board[new_c_x-1][new_c_y] != " ":
                        continue
                movement(update_board, board, checker,
new_c_x, init_c_x, new_c_y, init_c_y, opp_list, king, add,
white_list, black_list, check)
                break
        else:               # general movement of pawn on
any other position on the board
            if passant:
                for i in range(-1, 2, 2):
                    if board[init_c_x+a][init_c_y+i] in
piece_list:         # if it can kill a piece
                        if new_c_x == init_c_x+a and new_c_y
== init_c_y+i:
                            movement(kill, board, checker,
new_c_x, init_c_x, new_c_y, init_c_y, opp_list, king, add,
white_list, black_list, check)
                        elif new_c_x == init_c_x+a and
new_c_y == new_c_y and board[new_c_x][new_c_y] == " ":

                            movement(update_board, board,
checker, new_c_x, init_c_x, new_c_y, init_c_y, opp_list,
king, add, white_list, black_list, check)
                            break
                    else:
                        if new_c_x == init_c_x+a and new_c_y
== new_c_y and board[new_c_x][new_c_y] == " ":
                            movement(update_board, board,
checker, new_c_x, init_c_x, new_c_y, init_c_y, opp_list,
king, add, white_list, black_list, check)
```

```
                    break

        if new_piece != opponent_king:
            if new_c_x == final_x:          # promotion of a
pawn if it reaches the end of the board
                init_sel, promote = selection, ""
                print("\n", "What would you like to promote
", kill_dict[selection], " (", new_u_coords, ") to?", sep =
"")
                while promote not in promotion_dict:
                    promote = input().title()
                board[new_c_x][new_c_y] =
promotion_dict[promote]
                selection = board[new_c_x][new_c_y]
                print_board(board)
                promotion_statement =  kill_dict[init_sel] +
" (" + new_u_coords[0:2] + ") has been promoted to " +
kill_dict[selection]
                print("\n", promotion_statement)
                with open ("moves.txt", "a") as c1:
                    c1.write(promotion_statement + "\n")

        if inloop == False:
            break
        print("Invalid move!")

def rook(board, print_board, init_c_x, init_c_y, black_list,
white_list, new_coords_choice, kill, movement,
straight_block):          # elephant
    global back, counter, white_check, black_check, inloop,
blockage
    counter_checker(counter, white_list, black_list)
    if counter % 2 == 0:
        king, add, check, opp_list = chr(9812), black_add,
white_check, black_list
    elif counter % 2 == 1:
        king, add, check, opp_list = chr(9818), white_add,
black_check, white_list
        inloop = True
    while inloop:
        new_coords_choice(y_dict)
        if back:
            back = False
            break

        if (board[new_c_x][new_c_y] == " " or
board[new_c_x][new_c_y] in piece_list) and (new_c_y ==
init_c_y or new_c_x == init_c_x):
            straight_block(board, new_c_x, init_c_x,
new_c_y, init_c_y)
            if blockage:
                print("Path blocked! Invalid move!")
                continue
            elif blockage == False:
                movement(kill, board, checker, new_c_x,
init_c_x, new_c_y, init_c_y, opp_list, king, add,
white_list, black_list, check)
        if inloop == False:
            break
        print("Invalid move!")
```

```
def knight(board, print_board, init_c_x, init_c_y,
black_list, white_list, new_coords_choice, kill, movement):
# ek do dhai
    global back, counter, white_check, black_check, inloop
    counter_checker(counter, white_list, black_list)
    if counter % 2 == 0:
        king, add, check, opp_list = chr(9812), black_add,
white_check, black_list
    elif counter % 2 == 1:
        king, add, check, opp_list = chr(9818), white_add,
black_check, white_list
    inloop = True
    while inloop:
        new_coords_choice(y_dict)
        if back:
            back, inloop = False, False

        for i in range(-2, 3, 4):
            if (board[new_c_x][new_c_y] == " " or
board[new_c_x][new_c_y] in piece_list) and ((new_c_x ==
init_c_x+i and (new_c_y == init_c_y+(i//2) or new_c_y ==
init_c_y-(i//2))) or ((new_c_x == init_c_x+(i//2) or new_c_x
== init_c_x-(i//2)) and new_c_y == init_c_y+i)):
                movement(kill, board, checker, new_c_x,
init_c_x, new_c_y, init_c_y, opp_list, king, add,
white_list, black_list, check)
        if inloop == False:
            break
        print("Invalid move!")

def bishop(board, print_board, init_c_x, init_c_y,
black_list, white_list, new_coords_choice, kill, movement,
diagonal_block, breaker):
    global back, counter, white_check, black_check, inloop,
blockage, add
    counter_checker(counter, white_list, black_list)
    if counter % 2 == 0:
        king, add, check, opp_list = chr(9812), black_add,
white_check, black_list
    elif counter % 2 == 1:
        king, add, check, opp_list = chr(9818), white_add,
black_check, white_list
    inloop = True
    while inloop:
        new_coords_choice(y_dict)
        change = abs(new_c_y - init_c_y)        # horizontal
distance covered. if horizontal distance == vertical
distance, move is valid
        if back:
            back = False
            break

        if (board[new_c_x][new_c_y] == " " or
board[new_c_x][new_c_y] in piece_list) and ((new_c_x ==
init_c_x+change or new_c_x == init_c_x-change) and (new_c_y
== init_c_y+change or new_c_y == init_c_y-change)):
            diagonal_block(board, new_c_x, init_c_x,
new_c_y, init_c_y, change)
            if blockage:
                print("Path blocked! Invalid move!")
                continue
```

```
            elif blockage == False:
                movement(kill, board, checker, new_c_x,
init_c_x, new_c_y, init_c_y, opp_list, king, add,
white_list, black_list, check)
        if inloop == False:
            break
        print("Invalid move!")

def queen(board, print_board, init_c_x, init_c_y,
black_list, white_list, new_coords_choice, kill, movement,
diagonal_block, straight_block, breaker):
    global back, counter, white_check, black_check, inloop,
blockage, add
    counter_checker(counter, white_list, black_list)
    inloop = True
    if counter % 2 == 0:
        king, add, check, opp_list = chr(9812), black_add,
white_check, black_list
    elif counter % 2 == 1:
        king, add, check, opp_list = chr(9818), white_add,
black_check, white_list
    while inloop:              # queen's movement is
basically a combination of rook's and bishop's, so code for
her is copy-paste of both their codes
        new_coords_choice(y_dict)
        change = abs(new_c_y - init_c_y)
        if back:
            back = False
            break

        if (board[new_c_x][new_c_y] == " " or
board[new_c_x][new_c_y] in piece_list) and (((new_c_x ==
init_c_x+change or new_c_x == init_c_x-change) and (new_c_y
== init_c_y+change or new_c_y == init_c_y+change)) or
(new_c_y == init_c_y or new_c_x == init_c_x)):
            if (new_c_x == init_c_x+change or new_c_x ==
init_c_x-change) and (new_c_y == init_c_y+change or new_c_y
== init_c_y+change):       # if queen is moving diagonally
                diagonal_block(board, new_c_x, init_c_x,
new_c_y, init_c_y, change)
            elif new_c_y == init_c_y or new_c_x == init_c_x:
# if queen is moving in a straight line
                straight_block(board, new_c_x, init_c_x,
new_c_y, init_c_y)
            if blockage:
                print("Path blocked! Invalid move!")
                continue
            elif blockage == False:
                movement(kill, board, checker, new_c_x,
init_c_x, new_c_y, init_c_y, opp_list, king, add,
white_list, black_list, check)
        if inloop == False:
            break
        print("Invalid move!")

def king(board, print_board, init_c_x, init_c_y, black_list,
white_list, new_coords_choice, kill, selection, checker,
movement):
    global white_castling_checker, black_castling_checker,
counter, back, white_check, black_check, inloop
    if counter % 2 == 0:
```

```
        opp_list, initial_x, castling_checker, add, king,
check = black_list, 8, white_castling_checker, black_add,
chr(9812), white_check
    elif counter % 2 == 1:
        opp_list, initial_x, castling_checker, add, king,
check = white_list, 1, black_castling_checker, white_add,
chr(9818), black_check
    inloop = True
    while inloop:
        new_coords_choice(y_dict)
        if back:
            back, inloop = False, False
            break

        if castling_checker:            # for castling
            for i in range (-2, 3, 4):
                if new_c_x == initial_x and init_c_y == 5
and new_c_y == 5+i:
                    if i == 2:
                        if board[new_c_x][6] == " " and
board[new_c_x][7] == " ":
                            board[new_c_x][7],
board[new_c_x][6] = selection, board[new_c_x][8]
                            original_1, original_2 =
board[new_c_x][init_c_y], board[new_c_x][8]
                            board[new_c_x][init_c_y],
board[new_c_x][8] = " ", " "
                            checker(board, opp_list, king,
add)
                            if selection in white_list:
                                check = white_check
                            elif selection in black_list:
                                check = black_check
                            if check:
                                print("Invalid move!")
                                selection, board[new_c_x][8]
= board[new_c_x][7], board[new_c_x][6]
                                board[new_c_x][init_c_y],
board[new_c_x][8] = original_1, original_2
                                inloop, counter = False,
counter - 1
                                break
                            else:
                                print("\n", "Castled!",
"\n", sep = "")
                                print_board(board)
                                inloop, castling_checker =
False, False
                    elif i == -2:
                        if board[new_c_x][2] == " " and
board[new_c_x][3] == " " and board[new_c_x][4] == " ":
                            board[new_c_x][2],
board[new_c_x][3], board[new_c_x][4] =
board[init_c_x][init_c_y], selection, board[new_c_x][1]
                            original_3, original_4 =
board[init_c_x][init_c_y], board[new_c_x][1]
                            board[init_c_x][init_c_y],
board[new_c_x][1] = " ", " "
                            checker(board, opp_list, king,
add)
                            if selection in white_list:
```

```
                                check = white_check
                            elif selection in black_list:
                                check = black_check
                            if check:
                                print("Invalid move!")
                                board[init_c_x][init_c_y],
selection, board[new_c_x][1] =  board[new_c_x][2],
board[new_c_x][3], board[new_c_x][4]
                                board[init_c_x][init_c_y],
board[new_c_x][1] = original_3, original_4
                                inloop, counter = False,
counter - 1
                                break
                            else:
                                print("\n", "Castled!",
"\n", sep = "")
                                print_board(board)
                                inloop, castling_checker =
False, False
        for i in range (-1, 2, 2):
            if (board[new_c_x][new_c_y] == " " or
board[new_c_x][new_c_y] in opp_list) and ((new_c_x ==
init_c_x and new_c_y == init_c_y+i) or (new_c_y == init_c_y
and new_c_x == init_c_x+i) or (new_c_x == init_c_x+i and
new_c_y == init_c_y+i) or (new_c_x == init_c_x-i and new_c_y
== init_c_y+i)):
                movement(kill, board, checker, new_c_x,
init_c_x, new_c_y, init_c_y, opp_list, king, add,
white_list, black_list, check)
        if inloop == False:
            break
        print("Invalid move!")

#
============================================================
=================
# interface
#
============================================================
=================

print("\n", "Welcome to CHESS!", "\n")
print_board(board)

while True:
    if king_breaker:
        break
    breaker, selection, trapped, init_u_coords, init_u_x,
init_u_y, init_c_x, init_c_y, moves_kill = False, "", True,
"0z", 0, "0", 0, 0, False
    checker(board, black_list, white_king, black_add)
# checks if white king is checked
    checker(board, white_list, black_king, white_add)
# checks if black king is checked
    if counter % 2 == 0:
        if printer:
            print("\n", "Player 1's turn:", sep = "")
            if white_check:
                print("The White King is under check!")
            if black_check:
                print("The Black King is under check!")
```

```
        piece_list, you, opponent, printer = white_list,
"Player 1", "Player 2", True
    elif counter % 2 == 1:
        if printer:
            print("\n", "Player 2's turn:", sep = "")
            if white_check:
                print("The White King is under check!")
            if black_check:
                print("The Black King is under check!")
        piece_list, you, opponent, printer = black_list,
"Player 2", "Player 1", True

    select(board, piece_list, y_dict, trapped, pawn, rook,
knight, bishop, queen, king, select)
    if breaker:
        break
    print("You have selected", kill_dict[selection])
    if selection == chr(9817) or selection == chr(9823):
        pawn(board, print_board, init_c_x, init_c_y,
black_list, new_coords_choice, update_board, y_dict,
white_list, kill, en_passant, white_promotion_dict,
black_promotion_dict, selection, king_breaker, movement)
    elif selection == chr(9814) or selection == chr(9820):
        rook(board, print_board, init_c_x, init_c_y,
black_list, white_list, new_coords_choice, kill, movement,
straight_block)
    elif selection == chr(9816) or selection == chr(9822):
        knight(board, print_board, init_c_x, init_c_y,
black_list, white_list, new_coords_choice, kill, movement)
    elif selection == chr(9815) or selection == chr(9821):
        bishop(board, print_board, init_c_x, init_c_y,
black_list, white_list, new_coords_choice, kill, movement,
diagonal_block)
    elif selection == chr(9813) or selection == chr(9819):
        queen(board, print_board, init_c_x, init_c_y,
black_list, white_list, new_coords_choice, kill, movement,
diagonal_block, straight_block, breaker)
    elif selection == chr(9812) or selection == chr(9818):
        king(board, print_board, init_c_x, init_c_y,
black_list, white_list, new_coords_choice, kill, selection,
checker, movement)

    if moves_kill == False:
        if new_u_coords.lower() != "back":
            statement = kill_dict[selection] + " moved from
" + init_u_coords[0:2] + " to " + new_u_coords[0:2] + "\n"
            with open ("moves.txt", "a") as c1:
                c1.write(statement)

    counter += 1

#
============================================================
=================
# Data Management
#
============================================================
=================

import pymysql as pm
```

```
transcript_choice = ""
while transcript_choice not in ["y", "n"]:
    transcript_choice = (input("Would you like to save a
transcript of this game (y/n): ")).lower()
if transcript_choice == "y":
    filename = " "
    print("Note: If the filename already exists, its
contents will be overwritten.")
    while " " in filename:
        filename = input("Please type what you would like to
call this game (no spaces, 15 chars max): ")  + ".txt"
    with open(filename, "w") as c2:
        with open("moves.txt", "r") as c1:
            transcript = c1.read()
            c1.seek(0)
            linecount = len(c1.readlines()) - 1
        c2.write(transcript)
        print(filename, "has been saved on your system
successfully.")

    if winner != "draw":
        connection = pm.connect(host = "localhost", user =
"root", database = "chess1", password = "rajSim#1873")
        cursor = connection.cursor()
        cursor.execute("select count(*) from leaderboard;")
        leaderboard_len = cursor.fetchall()[0][0]
        cursor.execute("select moves from leaderboard;")
        try:
            top_moves = cursor.fetchall()[0]
            top_move = max(top_moves)
        except IndexError:
            top_move = linecount + 1
        if linecount < top_move:
            if leaderboard_len < 10:
                p1, p2 = input("Enter Player 1's name (12
chars max): "), input("Enter Player 2's name (12 chars max):
")
                if winner == "Player 1":
                    winner = p1
                else:
                    winner = p2
                cursor.execute("insert into leaderboard
values('{}', '{}', '{}', '{}',
{});".format(filename.rstrip(".txt"), p1, p2, winner,
linecount))
                connection.commit()
            else:
                cursor.execute("delete from leaderboard
where Moves = '{}';".format(top_move))
                connection.commit()
                cursor.execute("insert into leaderboard
values('{}', '{}', '{}', '{}',
{});".format(filename.rstrip(".txt"), p1, p2, winner,
linecount))
            print("This game has been added to the top 10
leaderboard!")
```

PROGRAM:
CHESS LEADERBOARD

```
import tkinter as tk, pymysql as pm
from tkinter.scrolledtext import ScrolledText
from tkinter import messagebox


connection = pm.connect(host = "localhost", user = "root", database =
"chess1", password = "rajSim#1873")
cursor = connection.cursor()


windowdimensions, windowtitle = "900x750+205+15", "Rajveer's Chess
Program"
landing_background, play_background, play_info_background,
gamerules_background = "Papaya Whip", "Light Sky Blue", "Pink",
"NavajoWhite2"
progrules_background, view_background, edit_background = "SeaGreen1",
"IndianRed1", "Thistle2"
p1_name, p2_name, game_name = "", "", ""
content1 = "Putting the opponent's king in a checkmate - a position
from which it is impossible to escape attack from your pieces."
content2 = '''White always moves first. Movement is required.
        With the exception of the knight, a piece may not move over
any of the other pieces.
        When a king is threatened with capture (but can protect
himself/escape), it´s called check.
        If a king is in check, then the player must eliminate the
threat and cannot leave the king in check.
        If a player isn't under check but has no legal moves, the
game is in stalemate and ends in a draw.
        Checkmate happens when a king is in check with no legal move
to escape. This ends the game.'''
content3 = '''- King can move 1 vacant square in any direction. It
may castle once per game.
        - Queen can move any number of vacant squares in any
direction.
        - Rook can move any number of vacant squares vertically or
horizontally. It also is moved while castling.
        - Bishop can move any number of vacant squares in any
diagonal direction.
        - Knight can move one square along any rank/file and then 2
perpendicularly. Its movement can also be viewed as an "L".
        - Pawns can move forward one vacant square. If not yet moved,
it may move 2 vacant squares.
        It cannot move backward. It kills diagonally forward. It can
also perform en passant and promotion.'''
content4 = '''- En Passant occurs when a pawn is moved 2 squares on
its initial movement.
        The opponent can take the moved pawn "en passant" as if it
had only moved one square.
```

```
        - If a pawn reaches the opponent's edge of the table, it  may
be promoted to a queen, rook, bishop or knight.
        - During the castling, the king moves two squares towards the
rook he intends to castle with,
        and the rook moves to the square through which the king
passed.
        Castling is only permissible if neither king nor rook
involved in castling may have
        moved from the original position and there are no pieces
between the rook and king.
        The king may not currently be in check, nor may the king pass
through or end up in a
        square that is under attack by an enemy piece.'''
content5 = '''- Enter coordinates of pieces by first mentioning the
row and then the column (without any spaces) and press 'return'.
        Capitalisation does not matter.
        For example: 2g, 4E, 8f, 1D

        - If a player's king is under checkmate, or if he wishes to
give up, he may type in "declare" where he is
        asked to enter the coordinates of the piece he wants to move,
after which he must confirm his declaration once again.
        If it is withdrawn, the game will continue.

        - If you wish to go back to selecting the initial coordinates
of the piece, type "back" where you are to type
        in the new coordinates.

        - If a game ends in a draw, any of the players can enter
"draw" where he's asked to enter the coordinates of
        the piece he wants to move. This call for draw will require a
confirmation from the other player.
        If the other player does not agree, the game will
continue.'''


landingwindow = tk.Tk()
landingwindow.geometry(windowdimensions)
landingwindow.title(windowtitle)
landingwindow.configure(bg = landing_background)


def play():
    playwindow = tk.Tk()
    playwindow.geometry(windowdimensions)
    playwindow.title(windowtitle)
    playwindow.configure(bg = play_background)


    def play_progrules():
        progruleswindow = tk.Tk()
        progruleswindow.geometry(windowdimensions)
        progruleswindow.title(windowtitle)
        progruleswindow.configure(bg = progrules_background)

        def progrules_back():
            progruleswindow.destroy()

        progrules_frame =          tk.Frame(progruleswindow, bg =
progrules_background, highlightbackground = "black",
highlightthickness = 12)
```

```
        progrules_padding1_label = tk.Label(progrules_frame, bg =
progrules_background, height = 1)
        progrules_title5_label =   tk.Label(progrules_frame, bg =
"gray25", fg = "white", font = ("Helvetica", 16, "bold"), text = "
Extra Rules: ")
        progrules_padding2_label = tk.Label(progrules_frame, bg =
progrules_background, height = 1)
        progrules_content5_label = tk.Label(progrules_frame, bg =
progrules_background, font = ("Helvetica", 12), text = content5)
        progrules_padding3_label = tk.Label(progrules_frame, bg =
progrules_background, height = 16)
        progrules_back_button =    tk.Button(progrules_frame, bg =
"black", fg = "white", font = ("Back to 1982", 14), text = "Back",
command = progrules_back)


        progrules_frame.pack(fill = "both", expand = "True")
        progrules_padding1_label.pack()
        progrules_title5_label.pack(fill = "x")
        progrules_padding2_label.pack()
        progrules_content5_label.pack()
        progrules_padding3_label.pack()
        progrules_back_button.pack()


    def play_gamerules():
        gameruleswindow = tk.Tk()
        gameruleswindow.geometry(windowdimensions)
        gameruleswindow.title(windowtitle)
        gameruleswindow.configure(bg = gamerules_background)

        def gamerules_back():
            gameruleswindow.destroy()


        gamerules_frame =           tk.Frame(gameruleswindow, bg =
gamerules_background, highlightbackground = "black",
highlightthickness = 12)
        gamnerules_padding1_label = tk.Label(gamerules_frame, height
= 1, bg = gamerules_background)
        gamerules_title1_label =    tk.Label(gamerules_frame, bg =
"gray25", fg = "white", font = ("Helvetica", 16, "bold"), text = "
Objective: ")
        gamerules_content1_label =  tk.Label(gamerules_frame, bg =
gamerules_background, fg = "black", font = ("Helvetica", 12), text =
content1)
        gamerules_padding2_label =  tk.Label(gamerules_frame, bg =
gamerules_background, height = 1)
        gamerules_title2_label =    tk.Label(gamerules_frame, bg =
"gray25", fg = "white", font = ("Helvetica", 16, "bold"), text = "
General Gameplay: ")
        gamerules_content2_label =  tk.Label(gamerules_frame, bg =
gamerules_background, fg = "black", font = ("Helvetica", 12), text =
content2)
        gamerules_padding3_label =  tk.Label(gamerules_frame, bg =
gamerules_background, height = 1)
        gamerules_title3_label =    tk.Label(gamerules_frame, bg =
"gray25", fg = "white", font = ("Helvetica", 16, "bold"), text = "
Piece Movement: ")
```

```
        gamerules_content3_label =  tk.Label(gamerules_frame, bg =
gamerules_background, fg = "black", font = ("Helvetica", 12), text =
content3)
        gamerules_padding4_label =  tk.Label(gamerules_frame, bg =
gamerules_background, height = 1)
        gamerules_title4_label =    tk.Label(gamerules_frame, bg =
"gray25", fg = "white", font = ("Helvetica", 16, "bold"), text = "
Additional Rules: ")
        gamerules_content4_label =  tk.Label(gamerules_frame, bg =
gamerules_background, fg = "black", font = ("Helvetica", 12), text =
content4)
        gamerules_padding5_label =  tk.Label(gamerules_frame, bg =
gamerules_background, height = 1)
        gamerules_back_button =     tk.Button(gamerules_frame, bg =
"black", fg = "white", font = ("Back to 1982", 14), text = "Back",
command = gamerules_back)


        gamerules_frame.pack(fill = "both", expand = True)
        gamnerules_padding1_label.pack()
        gamerules_title1_label.pack(fill = "x")
        gamerules_content1_label.pack()
        gamerules_padding2_label.pack()
        gamerules_title2_label.pack(fill = "x")
        gamerules_content2_label.pack()
        gamerules_padding3_label.pack()
        gamerules_title3_label.pack(fill = "x")
        gamerules_content3_label.pack()
        gamerules_padding4_label.pack()
        gamerules_title4_label.pack(fill = "x")
        gamerules_content4_label.pack()
        gamerules_padding5_label.pack()
        gamerules_back_button.pack()

    def play_back():
        playwindow.destroy()

    transcript_file = game_name  + ".txt"
    with open(transcript_file, "r") as m1:
        transcript = m1.read()
        m1.seek(0, 0)
        transcript_lines = m1.readlines()
    w1, b1, w2, b2 = 0, 0, 0, 0
    for i in transcript_lines:
        words = i.split()
        if words[0] == "White" and words[2] == "moved":
            w1 += 1
        elif words[0] == "Black" and words[2] == "moved":
            b1 += 1
        elif words[0] == "White" and words[3] == "killed":
            b2 += 1
        elif words[0] == "Black" and words[3] == "killed":
            w2 += 1
    cursor.execute("Select Winner from Leaderboard where Game_Name =
'{}';".format(game_name))
    winner_name = cursor.fetchall()[0][0]

    play_frame =                tk.Frame(playwindow, bg =
play_background, highlightbackground = "black", highlightthickness =
12)
```

```
    play_padding3_label =       tk.Label(play_frame, bg =
play_background, height = 1)
    play_name_label =           tk.Label(play_frame, text = p1_name+"
v. "+p2_name, bg = play_background, fg = "black", font = ("Back to
1982", 16))
    play_padding1_label =       tk.Label(play_frame, bg =
play_background, text = "     ")
    text_transcript =           ScrolledText(play_frame, height = 40,
width = 55)
    play_padding2_label =       tk.Label(play_frame, bg =
play_background, text = "     ")
    play_info_frame =           tk.Frame(play_frame, bg =
play_info_background, height = 645, width = 350)
    play_rules_frame =          tk.Frame(play_info_frame, bg =
play_info_background)
    play_padding5_label =       tk.Label(play_info_frame, bg =
play_info_background, height = 1)
    play_padding6_label =       tk.Label(play_rules_frame, bg =
play_info_background, text = "  ")
    play_progrules_button =     tk.Button(play_rules_frame, bg =
"black", fg = "white", text = " Program Rules ", font = ("Helvetica",
14, "bold"), command = play_progrules)
    play_padding4_label =       tk.Label(play_rules_frame, bg =
play_info_background, text = "  ")
    play_gamerules_button =     tk.Button(play_rules_frame, bg =
"black", fg = "white", text = "   Chess Rules   ", font =
("Helvetica", 14, "bold"), command = play_gamerules)
    play_padding7_label =       tk.Label(play_rules_frame, bg =
play_info_background, text = "  ")
    play_padding8_label =       tk.Label(play_info_frame, bg =
play_info_background, height = 1)
    play_winner_frame =         tk.Frame(play_info_frame, bg =
"dodger blue")
    play_win_label =            tk.Label(play_winner_frame, text =
"Winner:", font = ("Back to 1982", 14, "bold"), bg = "dodger blue",
fg = "white")
    play_winner_label =         tk.Label(play_winner_frame, text =
winner_name, font = ("Helvetica", 12, "bold"), bg = "dodger blue", fg
= "white")
    play_padding9_label =       tk.Label(play_info_frame, bg =
play_info_background, height = 7)
    play_moves_frame =          tk.Frame(play_info_frame, bg =
play_info_background)
    play_p1_moves_label =       tk.Label(play_moves_frame, text =
p1_name+"'s Moves: ", font = ("Helvetica", 14, "bold"), bg =
play_info_background, fg = "black")
    play_p1_movescount_label =  tk.Label(play_moves_frame, text = w1,
font = ("Helveteica", 14), bg = play_info_background, fg = "black")
    play_p2_moves_label =       tk.Label(play_moves_frame, text =
p2_name+"'s Moves: ", font = ("Helvetica", 14, "bold"), bg =
play_info_background, fg = "black")
    play_p2_movescount_label =  tk.Label(play_moves_frame, text = b1,
font = ("Helveteica", 14), bg = play_info_background, fg = "black")
    play_padding10_label =      tk.Label(play_info_frame, bg =
play_info_background, height = 4)
    play_pieces_frame =         tk.Frame(play_info_frame, bg =
play_info_background)
    play_p1_pieces_label =      tk.Label(play_pieces_frame, bg =
play_info_background, fg = "black", text = p1_name+"'s Pieces Killed:
", font = ("Helvetica", 14, "bold"))
```

```
    play_p1_piecescount_label = tk.Label(play_pieces_frame, bg =
play_info_background, fg = "black", text = w2, font = ("Helvetica",
14))
    play_p2_pieces_label =      tk.Label(play_pieces_frame, bg =
play_info_background, fg = "black", text = p2_name+"'s Pieces Killed:
", font = ("Helvetica", 14, "bold"))
    play_p2_piecescount_label = tk.Label(play_pieces_frame, bg =
play_info_background, fg = "black", text = b2, font = ("Helvetica",
14))
    play_padding11_label =      tk.Label(play_info_frame, bg =
play_info_background, height = 7)
    play_padding14_label =      tk.Label(play_info_frame, bg =
play_info_background, height = 1)
    play_quit_button =          tk.Button(play_info_frame, bg =
"black", fg = "white", font = ("Back to 1982", 13), text = " Back ",
command = play_back)
    play_padding15_label =      tk.Label(play_info_frame, bg =
play_info_background, height = 1)


    play_frame.pack(fill = "both", expand = True)
    play_padding3_label.pack()
    play_name_label.pack()
    play_padding1_label.pack(side = "left")
    text_transcript.pack(side = "left")
    play_padding2_label.pack(side = "right")
    play_info_frame.pack(side = "right")
    play_padding5_label.pack(side = "top")
    play_rules_frame.pack(side = "top")
    play_padding6_label.grid(row = 0, column = 0)
    play_progrules_button.grid(row = 0, column = 1)
    play_padding4_label.grid(row = 0, column = 2)
    play_gamerules_button.grid(row = 0, column = 3)
    play_padding7_label.grid(row = 0, column = 4)
    play_padding8_label.pack()
    play_winner_frame.pack(fill = "x", expand = True)
    play_win_label.pack(pady = 5)
    play_winner_label.pack()
    play_padding9_label.pack()
    play_moves_frame.pack()
    play_p1_moves_label.grid(row = 0, column = 0)
    play_p1_movescount_label.grid(row = 0, column = 1)
    play_p2_moves_label.grid(row = 1, column = 0)
    play_p2_movescount_label.grid(row = 1, column = 1)
    play_padding10_label.pack()
    play_pieces_frame.pack()
    play_p1_pieces_label.grid(row = 0, column = 0)
    play_p1_piecescount_label.grid(row = 0, column = 1)
    play_p2_pieces_label.grid(row = 1, column = 0)
    play_p2_piecescount_label.grid(row = 1, column = 1)
    play_padding11_label.pack()
    play_padding14_label.pack()
    play_quit_button.pack()
    play_padding15_label.pack()



    text_transcript.insert(tk.END, transcript)
    text_transcript.configure(state = "disabled")
```

```
def landing_quit():
    landingwindow.destroy()

def landing_leaderboard():
    def view_leaderboard():
        viewwindow = tk.Tk()
        viewwindow.geometry(windowdimensions)
        viewwindow.title(windowtitle)
        viewwindow.configure(bg = view_background)

        def view_back():
            viewwindow.destroy()

        cursor.execute("Select * from leaderboard order by moves
limit 0, 10")
        leaderdata = cursor.fetchall()

        view_frame =          tk.Frame(viewwindow, bg =
view_background, highlightbackground = "black", highlightthickness =
12)
        view_padding1_label = tk.Label(view_frame, bg =
view_background, height = 1)
        view_title_label =    tk.Label(view_frame, bg =
view_background, text = " Top 10 Leaderboard ", fg = "black", font =
("Helvetica", 20, "bold"))
        view_padding2_label = tk.Label(view_frame, bg =
view_background, height = 5)
        view_padding3_label = tk.Label(view_frame, bg =
view_background, height = 6)
        view_header_label =   tk.Label(view_frame, bg =
view_background, text = "     Rank     Game Name          Player1
Player2               Winner        Moves to Win     ", fg = "black",
font = ("Helvetica", 14, "bold"))
        view_board_frame =    tk.Label(view_frame, bg = "gray27",
highlightbackground = "black", highlightthickness = 3)
        view_back_button =    tk.Button(view_frame, bg = "black", fg
= "white", text = "Back", font = ("Back to 1982", 14), command =
view_back)

        view_frame.pack(fill = "both", expand = True)
        view_padding1_label.pack()
        view_title_label.pack()
        view_padding3_label.pack()
        view_header_label.pack()
        view_board_frame.pack()
        view_padding2_label.pack()
        view_back_button.pack()

        i = 0
        for rec in leaderdata:
            for j in range (len(leaderdata[0])):
                block_entry = tk.Entry(view_board_frame, width = 12,
fg = "black", bg = "white", font = ("Helvetica", 14))
                block_entry.grid(row = i, column = j+1, pady = 3,
padx = 1)
                block_entry.insert(tk.END, " " + str(rec[j]))
                block_entry.configure(state = "disabled")
            rank_entry =  tk.Entry(view_board_frame, width = 5, fg =
"black", bg = "white", font = ("Helvetica", 14))
```

```
            rank_entry.grid(row = i, column = 0, pady = 3, padx = 1)
            rank_entry.insert(tk.END, " " +str(i+1) +" ")
            rank_entry.configure(state = "disabled")
            i += 1

    def edit_leaderboard():
        editwindow = tk.Tk()
        editwindow.geometry(windowdimensions)
        editwindow.title(windowtitle)
        editwindow.configure(bg = edit_background)

        def edit_back():
            editwindow.destroy()

        def show_rank():
            cursor.execute("select Player1, Player2 from leaderboard
where Game_Name = '{}'".format(edit_rank.get()))
            show_rank_info = cursor.fetchall()
            edit_show_rank_label.configure(text = show_rank_info)

        def delete_record():
            if edit_rank.get() == "":
                messagebox.showinfo("Error", "Please select a Game to
operate on.")
            else:
                cursor.execute("delete from leaderboard where
Game_Name = '{}'".format(edit_rank.get()))
                connection.commit()
                messagebox.showinfo("Success", "The Record has been
deleted from the Leaderboard.")

        def edit_record():
            if edit_rank.get() == "":
                messagebox.showinfo("Error", "Please select a Game to
operate on.")
            else:
                def edit_record_back():
                    edit_attribute_label.forget()
                    edit_padding10_label.forget()
                    edit_attribute_frame.forget()
                    edit_padding7_label.forget()
                    edit_change_label.forget()
                    edit_padding8_label.forget()
                    edit_change_entry.forget()
                    edit_padding9_label.forget()
                    edit_submit_button.forget()
                    edit_padding2_label.forget()
                    edit_record_back_button.forget()
                    edit_padding5_label.pack()
                    edit_action_label.pack()
                    edit_padding6_label.pack()
                    edit_action1_button.pack()
                    edit_padding7_label.pack()
                    edit_action2_button.pack()
                    edit_padding2_label.pack()
                    edit_back_button.pack()

                edit_attribute, attribute_list =
tk.StringVar(editwindow), ["Player1", "Player2"]

                def edit_submit():
```

```
                    if edit_attribute.get() == "":
                        messagebox.showinfo("Error", "Please select a
Player Name to edit.")
                    elif edit_change_entry.get() == "":
                        messagebox.showinfo("Error", "Please enter a
value.")
                    else:
                            if len(edit_change_entry.get()) > 12:
                                messagebox.showinfo("Error", "Length
of Player Name cannot exceed 12 characters.")
                            else:
                                cursor.execute("Select Winner from
Leaderboard where Game_Name = '{}'".format(edit_rank.get()))
                                winner_name = cursor.fetchall()[0][0]
                                cursor.execute("Select {} from
leaderboard where Game_Name = '{}'".format(edit_attribute.get(),
edit_rank.get()))
                                player_name = cursor.fetchall()[0][0]
                                if winner_name == player_name:
                                    cursor.execute("update
leaderboard set winner = '{}' where Game_Name =
'{}'".format(edit_change_entry.get(), edit_rank.get()))
                                    connection.commit()
                                cursor.execute("update leaderboard
set {} = '{}' where Game_Name = '{}'".format(edit_attribute.get(),
edit_change_entry.get(), edit_rank.get()))
                                connection.commit()
                                messagebox.showinfo("Success",
"Leaderboard edited successfully. See the updated records in the
'View Leaderboard' tab.")

                edit_action_label.forget()
                edit_padding6_label.forget()
                edit_action1_button.forget()
                edit_padding7_label.forget()
                edit_action2_button.forget()
                edit_padding2_label.forget()
                edit_back_button.forget()

                edit_attribute_label =    tk.Label(edit_frame, bg =
edit_background, fg = "black", text = "Select an Attribute to Edit:",
font = ("Helvetica", 16, "bold"))
                edit_attribute_frame =    tk.Frame(edit_frame, bg =
edit_background)
                edit_change_label =       tk.Label(edit_frame, bg =
edit_background, fg = "black", text = "Enter the Updated Value:",
font = ("Helvetica", 16, "bold"))
                edit_padding8_label =     tk.Label(edit_frame, bg =
edit_background, height = 1)
                edit_change_entry =       tk.Entry(edit_frame,
highlightbackground = edit_background, width = 34)
                edit_padding9_label =     tk.Label(edit_frame, bg =
edit_background, height = 2)
                edit_submit_button =      tk.Button(edit_frame, bg =
"white", fg = "black", text = "Submit Changes", font = ("Back to
1982", 18), command = edit_submit)
                edit_padding10_label =    tk.Label(edit_frame, bg =
edit_background, height = 1)
                edit_record_back_button = tk.Button(edit_frame, bg =
"black", fg = "white", text = "Back", font = ("Back to 1982", 14),
command = edit_record_back)
```

```
                for i in attribute_list:
                        edit_attribute_radio =
tk.Radiobutton(edit_attribute_frame, text = " "+i+" ", var =
edit_attribute, value = i, indicatoron = 0, font = ("Helvetica", 10,
"bold"), width = 17)
                        edit_attribute_radio.grid(row = 0, column =
attribute_list.index(i))

                edit_attribute_label.pack()
                edit_padding10_label.pack()
                edit_attribute_frame.pack()
                edit_padding7_label.pack()
                edit_change_label.pack()
                edit_padding8_label.pack()
                edit_change_entry.pack()
                edit_padding9_label.pack()
                edit_submit_button.pack()
                edit_padding2_label.pack()
                edit_record_back_button.pack()

        edit_frame =            tk.Frame(editwindow, bg =
edit_background, highlightbackground = "black", highlightthickness =
12)
        edit_padding1_label =   tk.Label(edit_frame, bg =
edit_background, height = 2)
        edit_rank_label =       tk.Label(edit_frame, bg =
edit_background, fg = "black", text = "Select a Game Name to Edit its
Record:", font = ("Helvetica", 16, "bold"))
        edit_padding3_label =   tk.Label(edit_frame, bg =
edit_background, height = 1)
        edit_rank_frame =       tk.Frame(edit_frame, bg =
edit_background)
        edit_padding4_label =   tk.Label(edit_frame, bg =
edit_background, height = 1)
        edit_show_rank_label = tk.Label(edit_frame, bg =
edit_background, fg = "black", text = "Nothing Selected", font =
("Helvetica", 12, "bold"))
        edit_padding5_label =   tk.Label(edit_frame, bg =
edit_background, height = 2)
        edit_action_label =     tk.Label(edit_frame, bg =
edit_background, fg = "black", text = "Select what you want to do:",
font = ("Helvetica", 16, "bold"))
        edit_padding6_label =   tk.Label(edit_frame, bg =
edit_background, height = 5)
        edit_action1_button =   tk.Button(edit_frame, text = " Edit
Record", font = ("Back to 1982", 16, "bold"), width = 17, bg =
"white", fg = "black", command = edit_record)
        edit_padding7_label =   tk.Label(edit_frame, bg =
edit_background, height = 1)
        edit_action2_button =   tk.Button(edit_frame, text = " Delete
Record", font = ("Back to 1982", 16, "bold"), width = 17, bg =
"white", fg = "black", command = delete_record)
        edit_padding2_label =   tk.Label(edit_frame, bg =
edit_background, height = 9)
        edit_back_button =      tk.Button(edit_frame, bg = "black", fg
= "white", text = "Back", font = ("Back to 1982", 14), command =
edit_back)

        cursor.execute("select Game_Name from leaderboard order by
moves;")
```

```
        gamename_list = cursor.fetchall()

        edit_rank = tk.StringVar(editwindow)

        for i in range(10):
            try:
                name = gamename_list[i][0]
                edit_rank_radio = tk.Radiobutton(edit_rank_frame,
text = " "+name+" ", variable = edit_rank, value = name, indicatoron
= 0, font = ("Helvetica", 10, "bold"), width = 17, command =
show_rank)
                if i <= 4:
                    edit_rank_radio.grid(row = 0, column = i)
                else:
                    edit_rank_radio.grid(row = 1, column = i-5)
            except Exception:
                pass

        edit_frame.pack(fill = "both", expand = True)
        edit_padding1_label.pack()
        edit_rank_label.pack()
        edit_padding3_label.pack()
        edit_rank_frame.pack()
        edit_padding4_label.pack()
        edit_show_rank_label.pack()
        edit_padding5_label.pack()
        edit_action_label.pack()
        edit_padding6_label.pack()
        edit_action1_button.pack()
        edit_padding7_label.pack()
        edit_action2_button.pack()
        edit_padding2_label.pack()
        edit_back_button.pack()

    def landing_leaderboard_back():
        landing_padding1_label.configure(height = 13)
        landing_padding3_label.forget()
        landing_padding4_label.forget()
        landing_padding5_label.forget()
        landing_leaderboard_back_button.forget()
        landing_leaderboard_edit_button.forget()
        landing_leaderboard_view_button.forget()
        landing_buttons_frame.pack()
        landing_leaderboard_button.grid(row = 0, column = 0)
        landing_padding2_label.grid(row = 0, column = 1)
        landing_play_button.grid(row = 0, column = 2)
        landing_padding3_label.pack()
        landing_quit_button.pack()

    landing_padding1_label.configure(height = 1)
    landing_buttons_frame.forget()
    landing_padding3_label.forget()
    landing_quit_button.forget()
    landing_padding4_label =          tk.Label(landing_frame, bg =
landing_background, height = 3)
    landing_leaderboard_view_button = tk.Button(landing_frame, bg =
"white", fg = "black", text = "View Leaderboard", font = ("Back to
1982", 18), command = view_leaderboard)
    landing_padding5_label =          tk.Label(landing_frame, bg =
landing_background, height = 2)
```

```
    landing_leaderboard_edit_button = tk.Button(landing_frame, bg =
"white", fg = "black", text = "Edit Leaderboard", font = ("Back to
1982", 18), command = edit_leaderboard)
    landing_leaderboard_back_button = tk.Button(landing_frame, bg =
"black", fg = "white", text = "Back", font = ("Back to 1982", 14),
command = landing_leaderboard_back)

    landing_padding4_label.pack()
    landing_leaderboard_view_button.pack()
    landing_padding5_label.pack()
    landing_leaderboard_edit_button.pack()
    landing_padding3_label.pack()
    landing_leaderboard_back_button.pack()

def landing_play():
    global p1_name, p2_name

    def landing_play_back():
        landing_padding1_label.configure(height = 13)
        landing_padding3_label.forget()
        landing_padding4_label.forget()
        landing_padding5_label.forget()
        landing_play_frame.forget()
        landing_start_game_button.forget()
        landing_play_back_button.forget()
        landing_buttons_frame.pack()
        landing_leaderboard_button.grid(row = 0, column = 0)
        landing_padding2_label.grid(row = 0, column = 1)
        landing_play_button.grid(row = 0, column = 2)
        landing_padding3_label.pack()
        landing_quit_button.pack()

    def submission():
        global p1_name, p2_name, game_name
        if view_rank.get() == "":
            messagebox.showinfo("Error", "Please select a game to
view.")
        else:
            cursor.execute("Select Player1, Player2 from Leaderboard
where Game_Name = '{}'".format(view_rank.get()))
            player_names = cursor.fetchall()[0]
            p1_name, p2_name, game_name = player_names[0],
player_names[1], view_rank.get()
            play()


    landing_padding1_label.configure(height = 1)
    landing_buttons_frame.forget()
    landing_padding3_label.forget()
    landing_quit_button.forget()
    landing_padding4_label =    tk.Label(landing_frame, bg =
landing_background, height = 3)
    landing_play_frame =        tk.Frame(landing_frame, bg =
landing_background)
    landing_padding5_label =    tk.Label(landing_frame, bg =
landing_background, height = 4)
    landing_start_game_button = tk.Button(landing_frame, bg =
"white", fg = "black", text = "    View Game    ", font = ("Back to
1982", 18), command = submission)
```

```
    landing_play_back_button =  tk.Button(landing_frame, bg =
"black", fg = "white", text = "Back", font = ("Back to 1982", 14),
command = landing_play_back)

    cursor.execute("select Game_Name from leaderboard order by
moves;")
    gamename_list = cursor.fetchall()
    view_rank = tk.StringVar(landingwindow)
    for i in range(10):
        try:
            name = gamename_list[i][0]
            edit_rank_radio = tk.Radiobutton(landing_play_frame, text
= " "+name+" ", variable = view_rank, value = name, indicatoron = 0,
font = ("Helvetica", 10, "bold"), width = 17)
            if i <= 4:
                edit_rank_radio.grid(row = 0, column = i)
            else:
                edit_rank_radio.grid(row = 1, column = i-5)
        except Exception:
            pass

    landing_padding4_label.pack()
    landing_play_frame.pack()
    landing_padding5_label.pack()
    landing_start_game_button.pack()
    landing_padding3_label.pack()
    landing_play_back_button.pack()


landing_frame =              tk.Frame(landingwindow, bg =
landing_background, highlightbackground = "black", highlightthickness
= 12)
landing_padding1_label =     tk.Label(landing_frame, bg =
landing_background, height = 13)
chess_logo_label =           tk.Label(landing_frame, text = "  -
CHESS  -  ", font = ("back to 1982", 68), bg = landing_background, fg
= "black")
chess_line_label =           tk.Label(landing_frame, text = "---",
font = ("back to 1982", 54), bg = landing_background, fg = "black")
landing_buttons_frame =      tk.Frame(landing_frame, bg =
landing_background)
landing_leaderboard_button = tk.Button(landing_buttons_frame, bg =
"white", fg = "black", text = "Leaderboard", font = ("Back to 1982",
18), command = landing_leaderboard)
landing_play_button =        tk.Button(landing_buttons_frame, bg =
"white", fg = "black", text = " View Games ", font = ("Back to 1982",
18), command = landing_play)
landing_padding2_label =     tk.Label(landing_buttons_frame, bg =
landing_background, text = "    ")
landing_padding3_label =     tk.Label(landing_frame, bg =
landing_background, height = 5)
landing_quit_button =        tk.Button(landing_frame, fg = "white",
bg = "black", text = "Quit", font = ("Back to 1982", 14), command =
landing_quit)

landing_frame.pack(fill = "both", expand = True)
landing_padding1_label.pack()
chess_logo_label.pack()
chess_line_label.pack()
landing_buttons_frame.pack()
landing_leaderboard_button.grid(row = 0, column = 0)
```

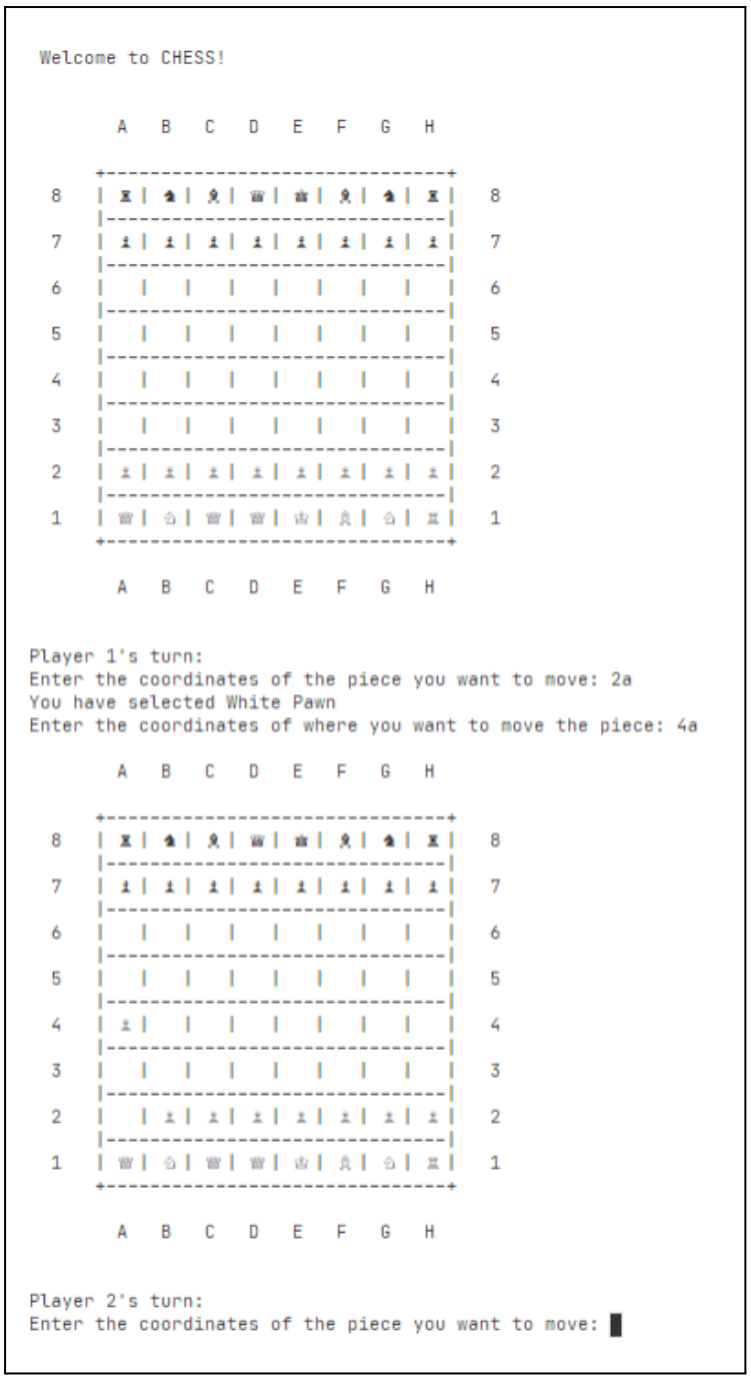
```
landing_padding2_label.grid(row = 0, column = 1)
landing_play_button.grid(row = 0, column = 2)
landing_padding3_label.pack()
landing_quit_button.pack()

landingwindow.mainloop()
```

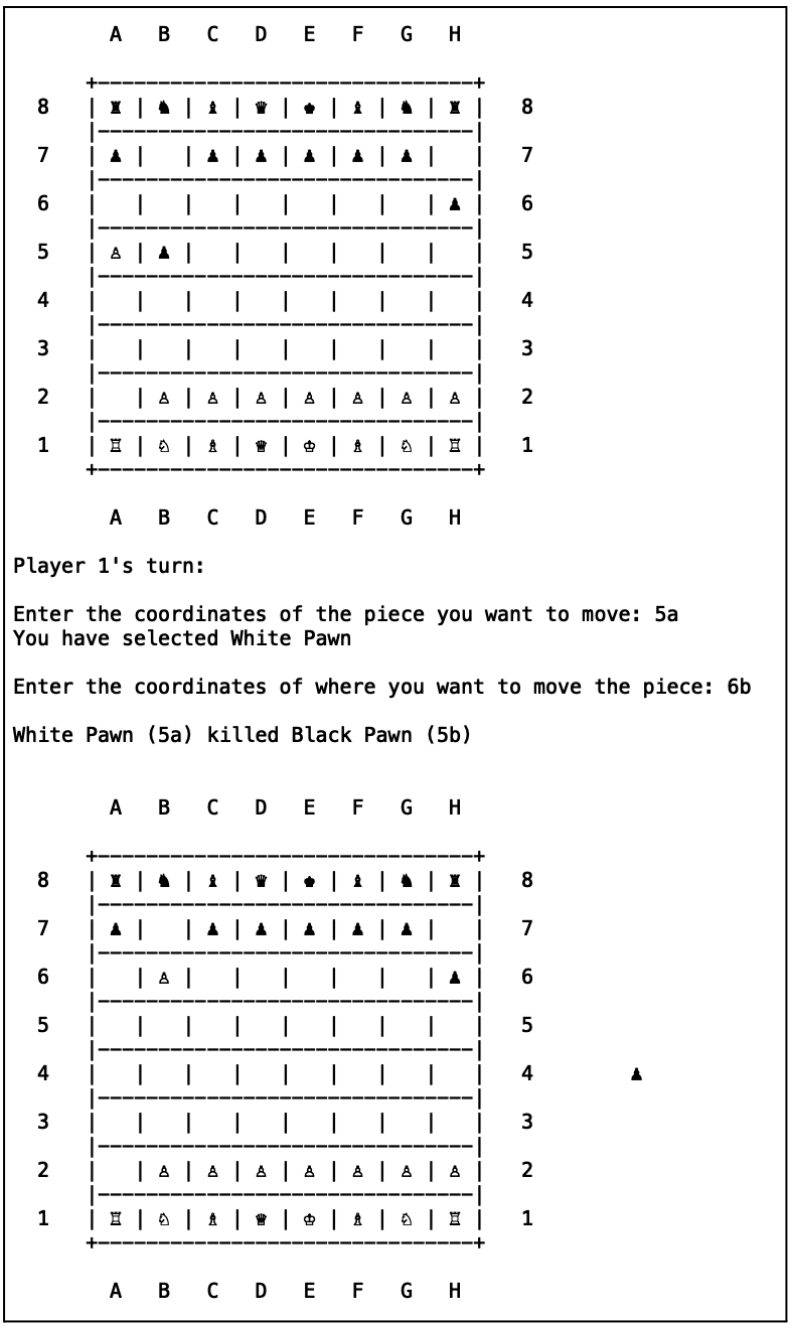
OUTPUT CONSOLE:

CHESS GAME

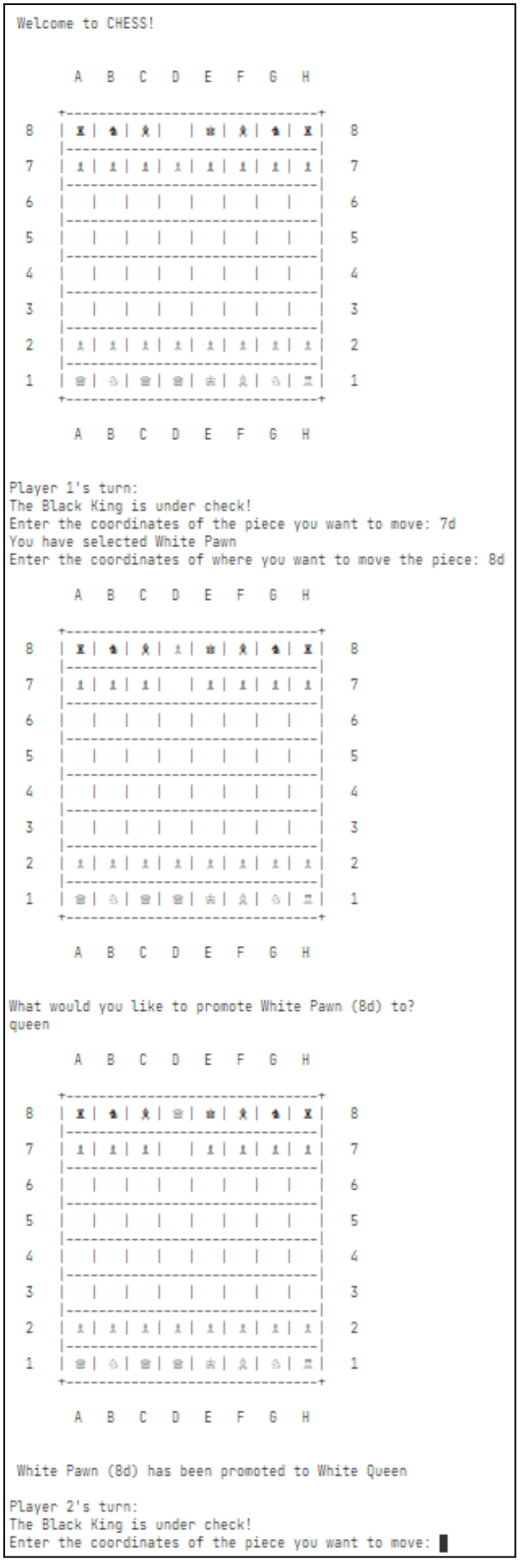
→ On starting the program, the users are greeted with the chess board. Starting with the white team, users can take turns moving their chess pieces throughout the board.



→ If a player is killed (as done using en passant in the picture below), the killed piece shows up on the side of the board.



→ If a player's pawn reaches the opposite end of the board, they have to promote the pawn to any other piece, such as a queen - as done in the picture below. Furthermore, it is also visible that once the pawn is promoted to a queen, the Black King is under check. Now, the Black team must make a move that immediately relieves their king of that situation.


Welcome to CHESS!
A B C D E F G H
8 ♜ ♞ ♝ ♛ ♚ ♝ ♞ ♜ 8
7 ♟ ♟ ♟ ♟ ♟ ♟ ♟ ♟ 7
6 6
5 5
4 4
3 3
2 ♙ ♙ ♙ ♙ ♙ ♙ ♙ ♙ 2
1 ♖ ♘ ♗ ♕ ♔ ♗ ♘ ♖ 1
A B C D E F G H
Player 1's turn:
The Black King is under check!
Enter the coordinates of the piece you want to move: 7d
You have selected White Pawn
Enter the coordinates of where you want to move the piece: 8d
A B C D E F G H
8 ♜ ♞ ♝ ♟ ♛ ♝ ♞ ♜ 8
7 ♟ ♟ ♟ ♟ ♟ ♟ ♟ 7
6 6
5 5
4 4
3 3
2 ♙ ♙ ♙ ♙ ♙ ♙ ♙ ♙ 2
1 ♖ ♘ ♗ ♕ ♔ ♗ ♘ ♖ 1
A B C D E F G H
What would you like to promote White Pawn (8d) to?
queen
A B C D E F G H
8 ♜ ♞ ♝ ♕ ♛ ♝ ♞ ♜ 8
7 ♟ ♟ ♟ ♟ ♟ ♟ ♟ 7
6 6
5 5
4 4
3 3
2 ♙ ♙ ♙ ♙ ♙ ♙ ♙ ♙ 2
1 ♖ ♘ ♗ ♕ ♔ ♗ ♘ ♖ 1
A B C D E F G H
White Pawn (8d) has been promoted to White Queen
Player 2's turn:
The Black King is under check!
Enter the coordinates of the piece you want to move:

→ If the player enters “draw” or “declare” where they’re asked to enter the initial coordinates, the game can be ended. They must do so if their king is in checkmate. Once a game ends, the user gets the option to save a transcript of the game. After doing this, their game - considering it did not end in a draw - is considered for position in the Top 10 Leaderboard.

```
Player 1's turn:

Enter the coordinates of the piece you want to move: draw
Player 1 is calling for a draw! Does PLayer 2 agree? (Type draw if yes):

draw
The game ended in a draw!
```

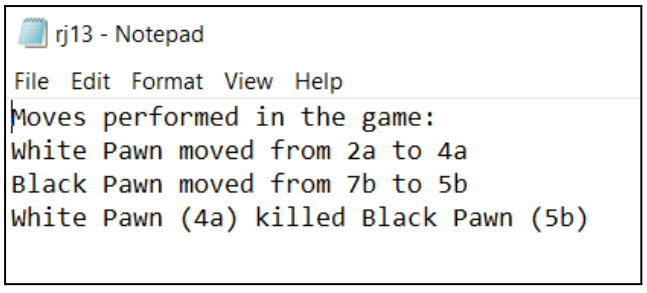
```
Player 2's turn:

Enter the coordinates of the piece you want to move: declare

Are you sure? (y/n): y
Player 2 declared defeat. Player 1 wins!
```

```
Would you like to save a transcript of this game (y/n): y
Note: If the filename already exists, its contents will be overwritten.
Please type what you would like to call this game (no spaces, 15 chars max): fast2
fast2.txt has been saved on your system successfully.
Enter Player 1's name (12 chars max): abhimanyu
Enter Player 2's name (12 chars max): rajveer
This game has been added to the top 10 leaderboard!
```

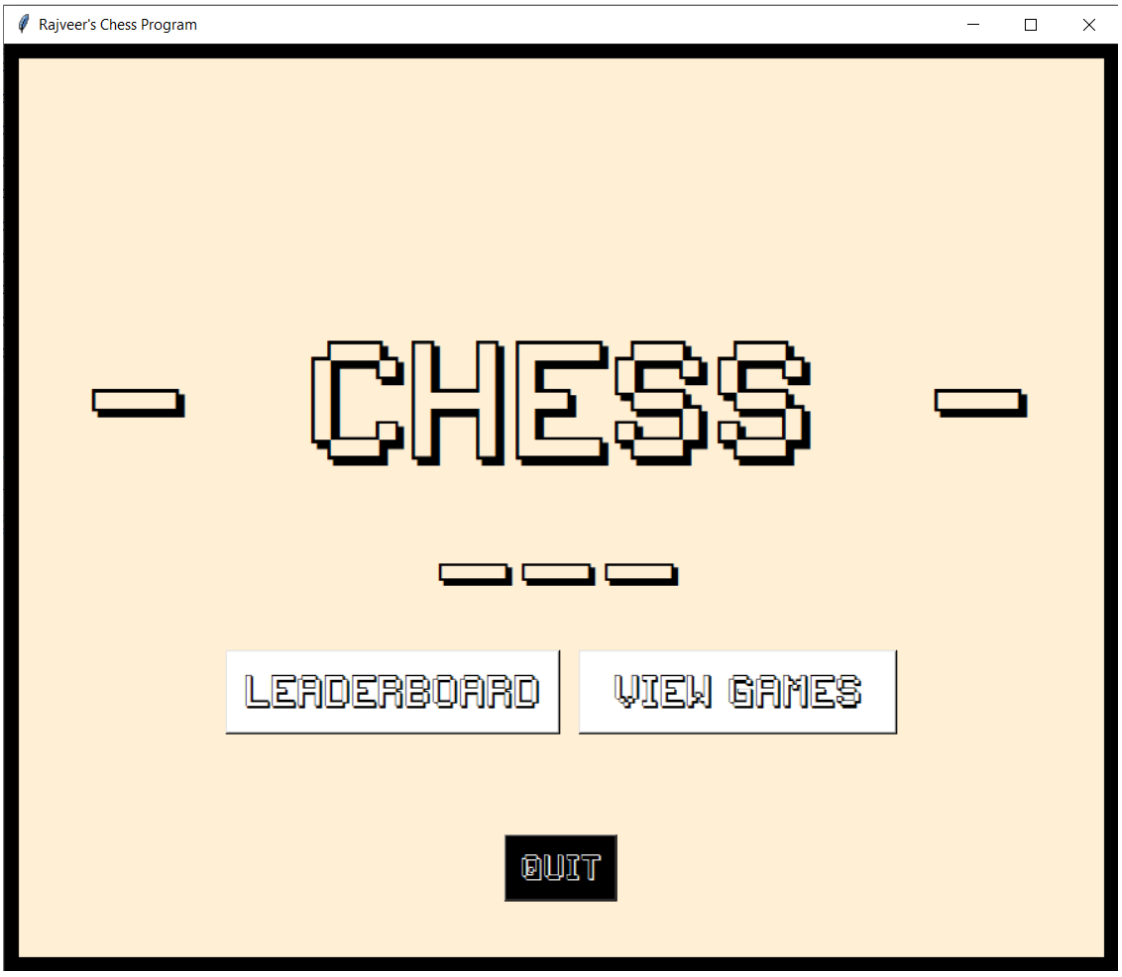
→ The transcript can be found on the users laptop as a .txt file.



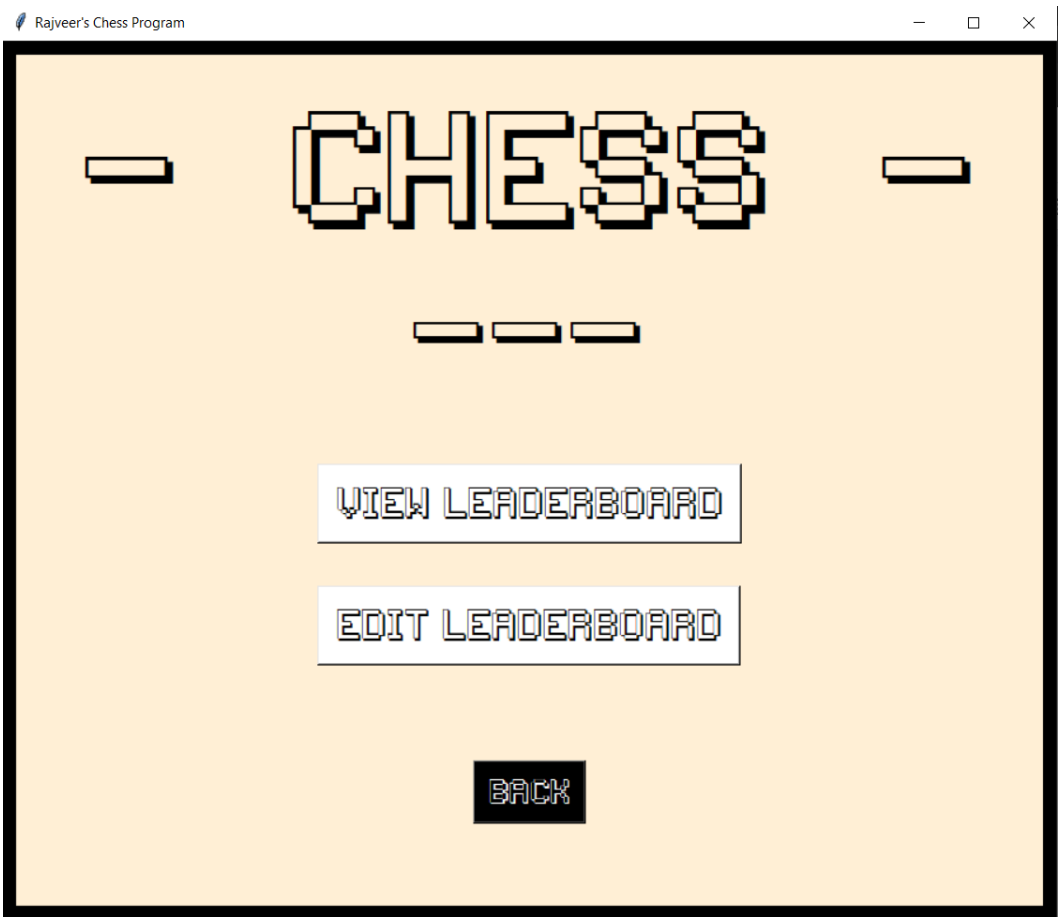
OUTPUT CONSOLE:

CHESS LEADERBOARD

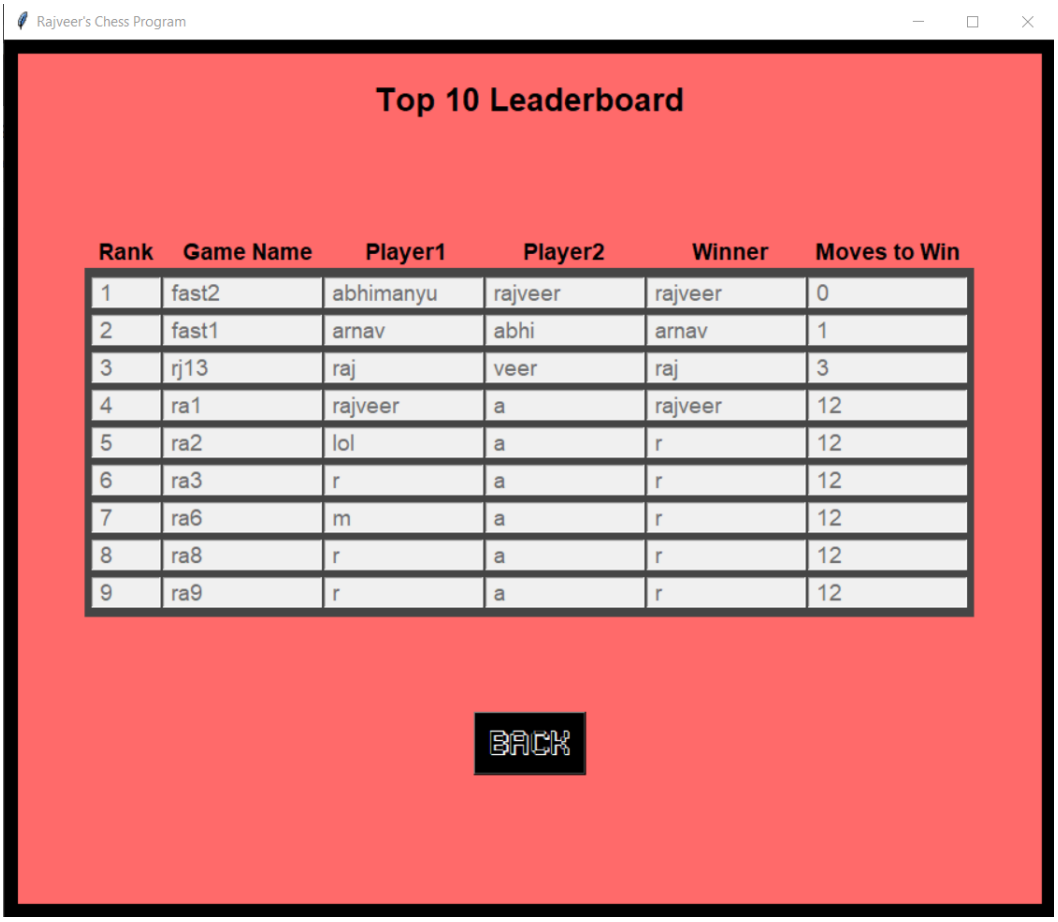
→ The application opens to a landing page which allows the user to either deal with the leaderboard or see details about games in the leaderboard.



→ Pressing the leaderboard button, the user is given the option to either edit or view the current leaderboard.



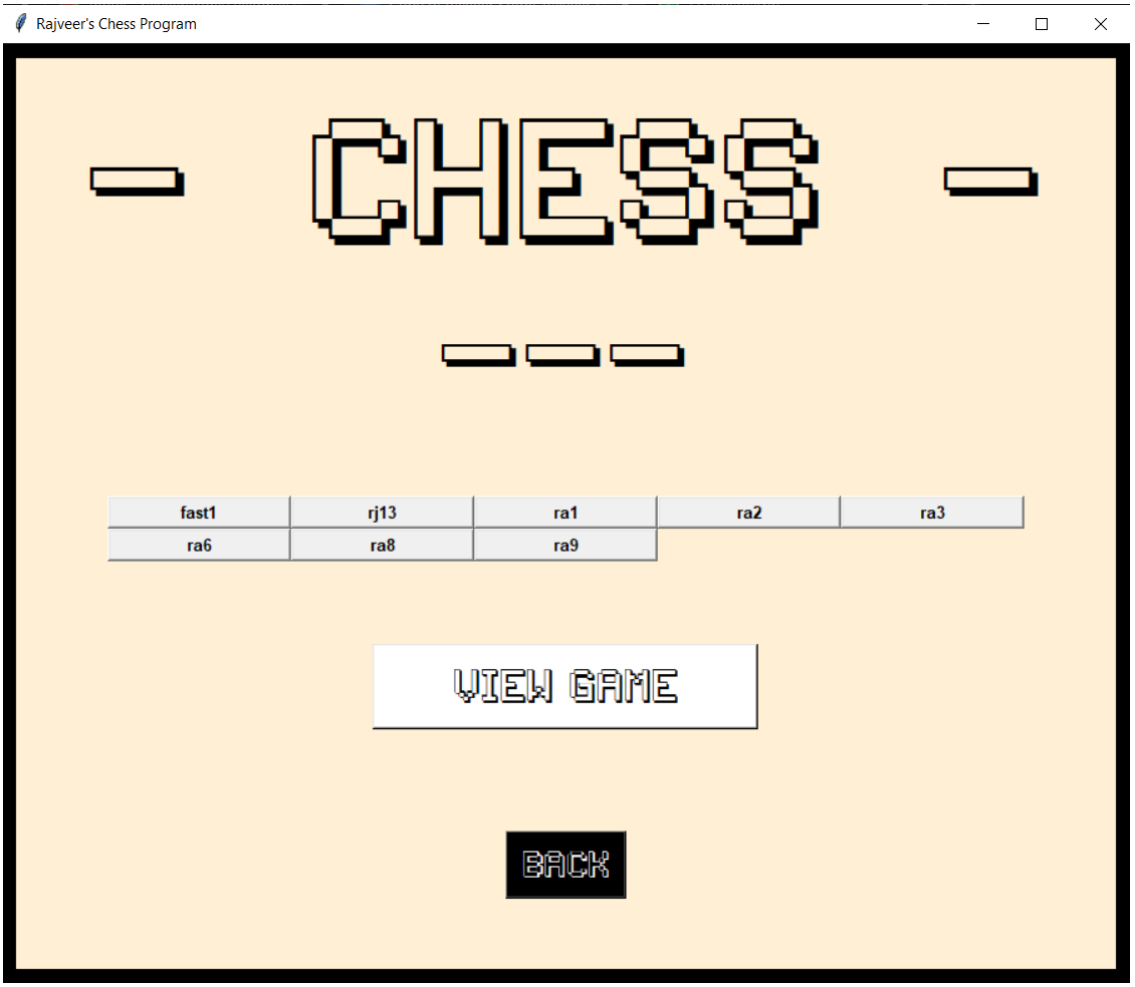
→ The button View Leaderboard opens the following page.



→ The Edit Leaderboard button opens the following page, where the user must choose one of the ten games in the leaderboard to perform operations on.

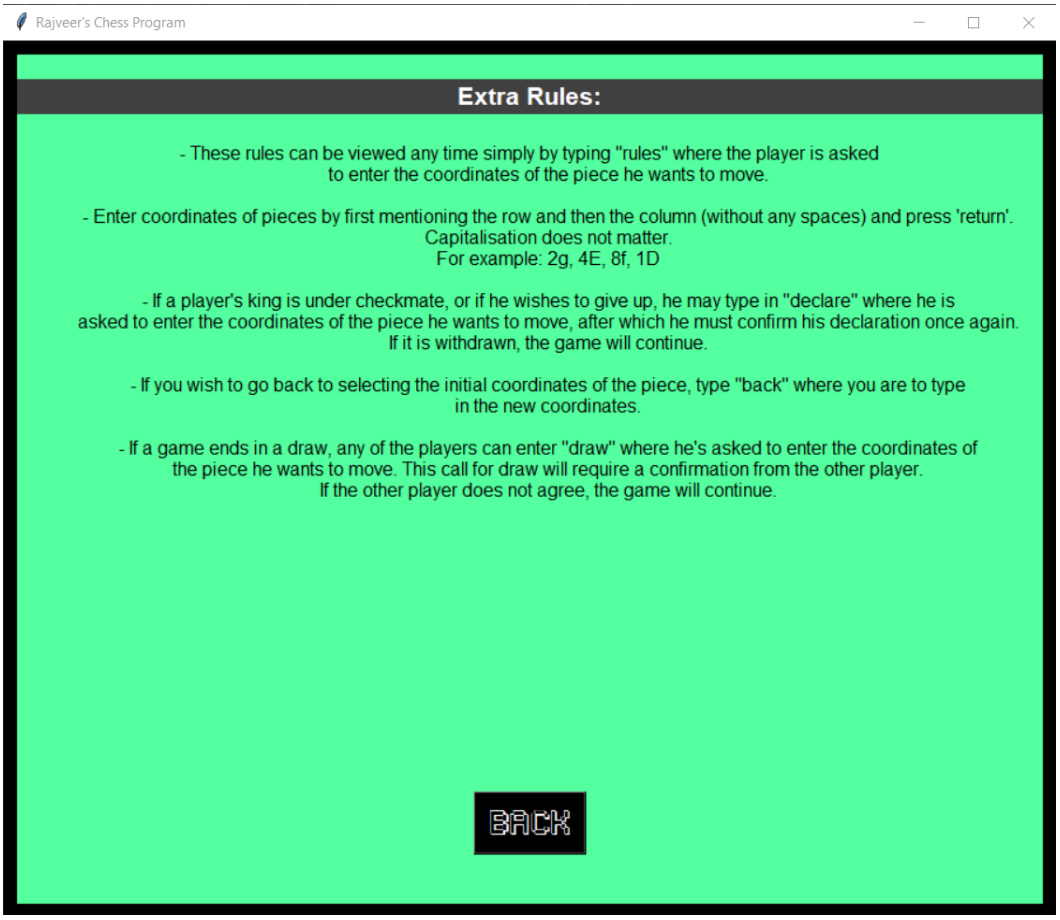

Rajveer's Chess Program
Select a Game Name to Edit its Record:
fast2
fast1
rj13
ra1
ra2
ra3
ra6
ra8
ra9
{abhimanyu rajveer}
Select what you want to do:
EDIT RECORD
DELETE RECORD
BACK

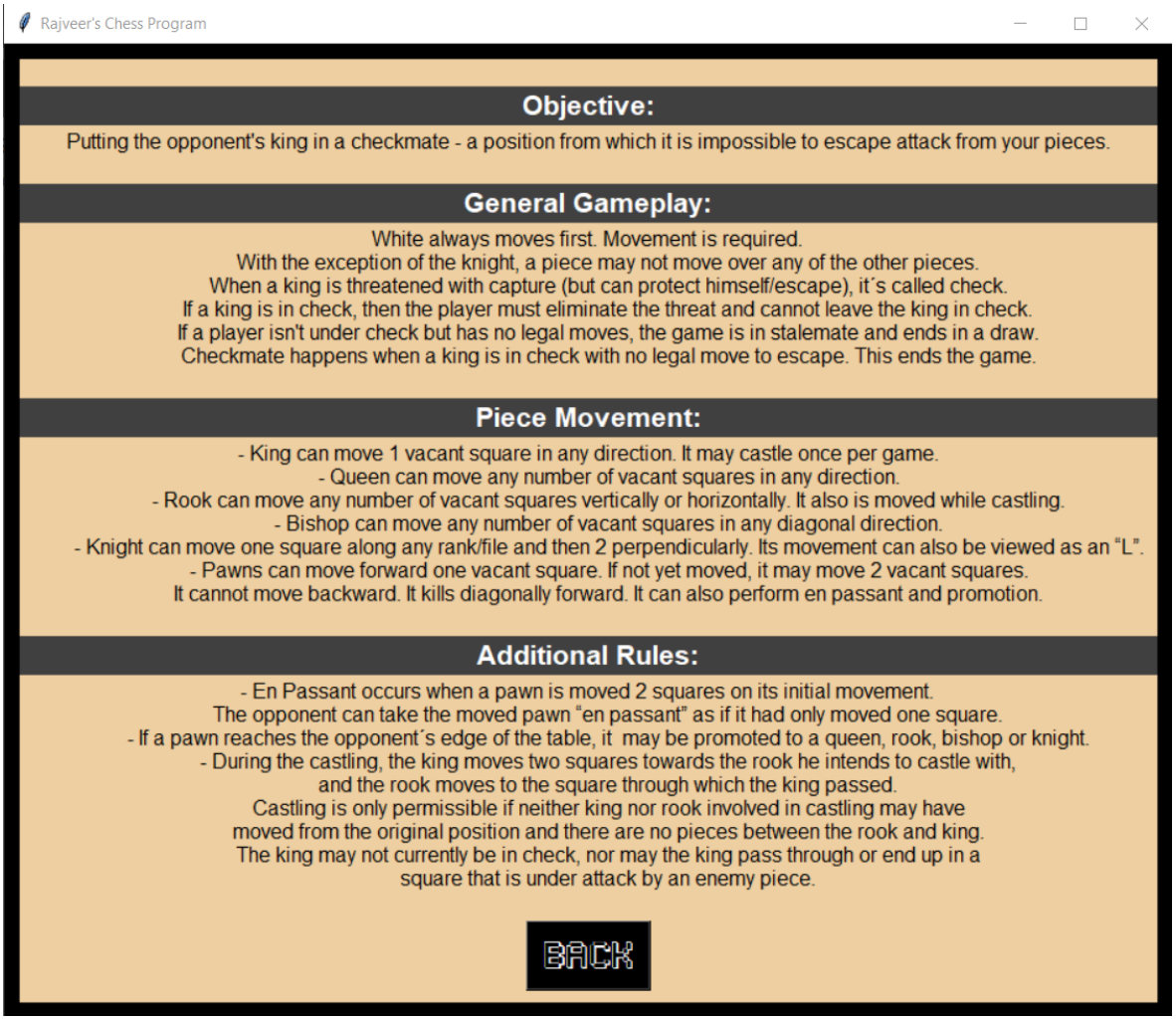
→ From the Landing Page, the View Leaderboard button brings the user to a page which asks them to choose a game to view the details of.



→ This page allows the user to see the transcript of the game in the panel on the left, and additional information on the right. Further, the player can also view necessary rules from here.


Rajveer's Chess Program
ARNAV V. ABHI
Moves performed in the game:
White Pawn moved from 2a to 4a
Program Rules
Chess Rules
WINNER:
arnav
arnav's Moves: 1
abhi's Moves: 0
arnav's Pieces Killed: 0
abhi's Pieces Killed: 0
BACK


Rajveer's Chess Program
Extra Rules:
- These rules can be viewed any time simply by typing "rules" where the player is asked to enter the coordinates of the piece he wants to move.
- Enter coordinates of pieces by first mentioning the row and then the column (without any spaces) and press 'return'. Capitalisation does not matter. For example: 2g, 4E, 8f, 1D
- If a player's king is under checkmate, or if he wishes to give up, he may type in "declare" where he is asked to enter the coordinates of the piece he wants to move, after which he must confirm his declaration once again. If it is withdrawn, the game will continue.
- If you wish to go back to selecting the initial coordinates of the piece, type "back" where you are to type in the new coordinates.
- If a game ends in a draw, any of the players can enter "draw" where he's asked to enter the coordinates of the piece he wants to move. This call for draw will require a confirmation from the other player. If the other player does not agree, the game will continue.
BACK


Rajveer's Chess Program
Objective:
Putting the opponent's king in a checkmate - a position from which it is impossible to escape attack from your pieces.
General Gameplay:
White always moves first. Movement is required.
With the exception of the knight, a piece may not move over any of the other pieces.
When a king is threatened with capture (but can protect himself/escape), it´s called check.
If a king is in check, then the player must eliminate the threat and cannot leave the king in check.
If a player isn't under check but has no legal moves, the game is in stalemate and ends in a draw.
Checkmate happens when a king is in check with no legal move to escape. This ends the game.
Piece Movement:
- King can move 1 vacant square in any direction. It may castle once per game.
- Queen can move any number of vacant squares in any direction.
- Rook can move any number of vacant squares vertically or horizontally. It also is moved while castling.
- Bishop can move any number of vacant squares in any diagonal direction.
- Knight can move one square along any rank/file and then 2 perpendicularly. Its movement can also be viewed as an "L".
- Pawns can move forward one vacant square. If not yet moved, it may move 2 vacant squares.
It cannot move backward. It kills diagonally forward. It can also perform en passant and promotion.
Additional Rules:
- En Passant occurs when a pawn is moved 2 squares on its initial movement.
The opponent can take the moved pawn "en passant" as if it had only moved one square.
- If a pawn reaches the opponent´s edge of the table, it may be promoted to a queen, rook, bishop or knight.
- During the castling, the king moves two squares towards the rook he intends to castle with,
and the rook moves to the square through which the king passed.
Castling is only permissible if neither king nor rook involved in castling may have
moved from the original position and there are no pieces between the rook and king.
The king may not currently be in check, nor may the king pass through or end up in a
square that is under attack by an enemy piece.
BACK

REFERENCES

- Computer Science with Python XII - Preeti Arora