CLASSIC CHESS

PROJECT REPORT

SRI SATHYA SAI INSTITIUTE OF HIGHER LEARNING, PRASHANTHI NILAYAM
(DEEMED TO BE UNIVERSITY)
DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE
BY,
SRIVATS RAMASWAMY
BCA-III 174421
SRI SATHYA SAI INSTITUTE OF HIGHER LEARNING, MUDDENAHALLI CAMPUS

CERTIFICATE

This is to certify that this project titled **Classic Chess** submitted by Srivats Ramaswamy, Department of Mathematics and Computer Science, Muddenahalli Campus is a bonafide record of the original work done under my/our supervision as a course requirement for the Degree of Bachelors of Computer Applications.

…….

Sri Udhaya Ravishankar,

Project Supervisor

…….

Countersigned By

Sri B.Venkatramana

Deputy Director

Sri Sathya Sai Institute of Higher Learning,

Muddenahalli Campus

Date: 20th March 2020

Place: Muddenahalli

Contents

Offering my report and project at the Divine Lotus Feet of our beloved Bhagawan Sri Sathya Sai Baba

Acknowledgements

I would like to thank **Sri Udhaya Ravi Shankar** sir for giving me his valuable insights and guidance throughout the project. If there are certain errors in the project, I (the developer) happen to be responsible for it.

I would like to thank **Mr. Albert Swiegart**. Albert Sweigart is a software developer in San Francisco, California. His first book, '**Invent Your Own Computer Games with Python**' can be read online at *http://inventwithpython.com*. **Making Games with Python & Pygame** is his second book which happens to be the book I have referred to complete my project. He is originally from Houston, Texas and also is a computer science graduate from *University of Texas*.

I would like to thank **Mr. Brandon Rhodes**. Brandon Rhodes is a consulting programmer who also teaches the Python language professionally for organizations that are adding the language to their tool set. He has spoken at PyOhio, PyGotham, national PyCon conferences in Canada, Ireland, and Poland; and at Django conferences in Portland, Wales, and Warsaw.

He has chaired the flagship PyCon North America conference in Portland in 2016–2017. He has authored the book **Foundations of Python Network Programming** which enabled me to learn about networking concepts and apply them in my project.

Last but not the least, I would like to thank **Bhagawan Sri Sathya Sai Baba** for his grace, guidance and support. **Without him nothing can ever be possible.**

Problem Statement

Building a multiplayer chess application. In this application the player can do as follows:

In this application, 2 players connect with each other on a **Local Area Network** and play the classic chess game. One of players acts as the **host(server)** and the other acts as the **guest(client)**. The guest is the given the choice to choose between black or white. The host gets the choice of whatever the guest chooses. Thus, both the players are connected and the game can be played.

Aim of The Project:

E-sports has been a growing movement all over the world. Games like **Counter Strike Global Offensive, DOTA 2, League of Legends** and many more games have come to the fore front and have given a chance for players all over the world to compete. Hence, this game was an attempt to bring about e-sports in this campus. Unlike other games, I believe chess can be an ethical choice considering the terms and conditions of **Sri Sathya Sai Institute of Higher Learning**. **Chess** is a simple and strategic game which helps in **the development of the brain in terms of logical thinking and analysis**. Hence, a **classic version of the chess** was what I chose as the game of choice to implement as the perfect game for competing amongst the students.

Implementation

This project was implemented using the following:

Python:

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL).

Why Python?

This project was done in Python for the following reasons:

Software Quality and ease of usage:

For many, Python's focus on readability, coherence, and software quality in general sets it apart from other tools in the scripting world. Python code is designed to be readable, and hence reusable and maintainable. Python has deep support for more advanced software reuse mechanisms, such as object-oriented programming (OOP).

Program Portability:

Most Python programs run unchanged on all major computer platforms. Porting Python code between Linux and Windows, for example, is usually just a matter of copying a script's code between machines. Moreover, Python offers multiple options for coding

portable graphical user interfaces, database access programs, web based systems, and more.

Support libraries:

Python comes with a large collection of prebuilt and portable functionality, known as the standard library. This library supports an array of application-level programming tasks, from text pattern matching to network scripting. In addition, Python can be extended with both home grown libraries and a vast collection of third-party application support software.

Developer Productivity:

Python boosts developer productivity many times beyond compiled or statically typed languages such as C, C++, and Java. Python code is typically one-third to one-fifth the size of equivalent C++ or Java code. Since Python does not take time to compile it, runs faster as it interprets the code directly.

Popularity:

Many famous organisations and institutions such as **Google, NSA, NASA, JPMorgan, Cisco** and many more organizations use Python language for implementation for most of their applications and other solutions. In terms of game development, **Electronic Arts** has used Python to code some parts of their renowned game called "**Battlefield"**. **Maya**, a graphical model designing software also provides Python API along with other APIs.

Implemented Modules*:*

1. **Pygame**:

Pygame is built on another game creation library called Simple DirectMedia Layer (SDL). SDL was written by Sam Lantinga while he was working for Loki Software (a now-defunct game company) to simplify the task of porting games from one platform to another. It provided a common way to create a display on multiple platforms as well as work with graphics and input devices. Because it was so simple to work with, it became very popular with game developers

when it was released in 1998, and has since been used for many hobby and commercial games.

2. **Socket**:

   Socket module used in order write and develop network based applications. The socket module is available by default as part of the Python languages standard libraries meaning the programmer does not have install this package manually.

   Socket can be used for implementing a wide variety of networking protocols such as UDP and TCP/IP and forms the standard basic module for any programmer to start learning about networking in Python.

Hardware Components

Memory: 7.7 GB

CPU:

- **Processor:** Intel Core i3-3220 CPU @3.30 GHz
- **Number of Cores:** 4
- **Graphics:** Intel Ivybridge Desktop
- **Architecture:** X-86_64
- **CPU op-modes:** 32-bit, 64-bit
- **Byte-Order:** Little Endian

Operating System:

- **OS Name:** Ubuntu 18.04.2 LTS
- **OS Type:** 64-bit
- **GNOME:** 3.28.2

Disk: 491.2 GB

Core Concepts

The core concepts that have been incorporated during the course of my project are as follows:

1. **2D Graphics (Image Rendering):**

   2D Graphics essentially deals with objects which can be represented in terms of 2 dimensional coordinates being the **x-axis** and the **y-axis** coordinates respectively.

   An object (which in 2D is an image) has 2 properties namely **width and height. Width** is represented by the **x-axis** and **height** is represented by the **y-axis** respectively. These coordinates are notations in order to map the **pixels** that comprises an object (2D image).

   Pixels is the standard unit when the developer deals with graphics whether 2D or 3D. Pixels are the tiny square dots on the screen which initially start as black but can be set to any colour. Pixel colours can be set using the RGB (Red Green Blue) standard in pygame. The values in RGB range from 0-255. The colour combination for black is (0,0,0) and for white is (255,255,255). When it comes to images the change in the pixel coordinates determines the change in the image position. The coordinate axis on a computer screen follows the 4^{th} quadrant of the coordinate axis where the top left corner of the screen is the position of the origin (0,0). The x-axis progresses towards the right and the y-axis progresses downwards.

   Each time the image moves from an initial coordinate position to another, it constitutes a **Frame**. **A sequence of 24 different frames is the minimum requirement for enabling the human eye to detect the motion of the image on the screen** and not just changes in position of the image. This series of frames to move an image from its initial position to a target position is called as **Image Rendering**.

Graphics more specifically rendering objects fall under 2 categories:

a. **Interactive:**

Graphics becomes interactive when the images or graphical objects can be rendered using user input which are called events. The Frame Rate is usually high to deliver the user a rich experience of the application making it dynamic. It is mostly used while developing video games.

b. **Non-Interactive:**

Non-Interactive graphics renders graphical objects as per a prewritten script. The application is static as no user input can be accepted. It generally renders minimum number of frames which is required to generate the smooth movement of the objects. It is widely used in making animated films and visual effects.

2. **UDP:**

UDP stands for **User Datagram Protocol.**

3. **Dictionaries:**

A dictionary is like a list, but more general. In a list, the index positions have to be integers; in a dictionary, the indices can be (almost) any type.

A dictionary can be thought of as a mapping between a set of indices (which are called keys) and a set of values. Each key maps to a value. The association of a key and a value is called a key-value pair or sometimes an item. Dictionaries in python can be declared as follows:

```
bca3@mdh116:~$ python3
Python 3.6.7 (default, Oct 22 2018, 11:32:17)
[GCC 8.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> my_dict={'name':'Srivats','job':'Student','age':20}
>>> my_dict['name']
'Srivats'
>>> my_dict['job']
'Student'
>>> my_dict['age']
20
```

We can see from the above example about the mapping of the key-value pair.

Dictionaries Vs Classes (Why dictionaries, not classes):

Dictionaries are a lot better than classes in terms of access time. Dictionary happens to be faster in terms of accessing data than accessing data which has to be referred using objects.

As proof for the above statement a screenshot has been inserted below which is from the site http://www.githubgist.com

Speed (Time) Tests

Accessing an existing value

Fastest: Accessing a dictionary value through the index operator

```
$ python3.2 -mtimeit -s"pnoun = {'nom': 'she', 'obl': 'her', 'pos_det': 'her', 'pos_pro': 'hers', 'reflex':

1000000 loops, best of 3: 0.419 usec per loop
```

Faster: Accessing an object field through the namespace operator

```
$ python3.2 -mtimeit -s"from gender import GenderPronouns; pnoun = GenderPronouns('she', 'her', 'her', 'hers

1000000 loops, best of 3: 0.446 usec per loop
```

Slower: Accessing a dictionary value through the get() method

```
$ python3.2 -mtimeit -s"pnoun = {'nom': 'she', 'obl': 'her', 'pos_det': 'her', 'pos_pro': 'hers', 'reflex':

1000000 loops, best of 3: 0.637 usec per loop
```

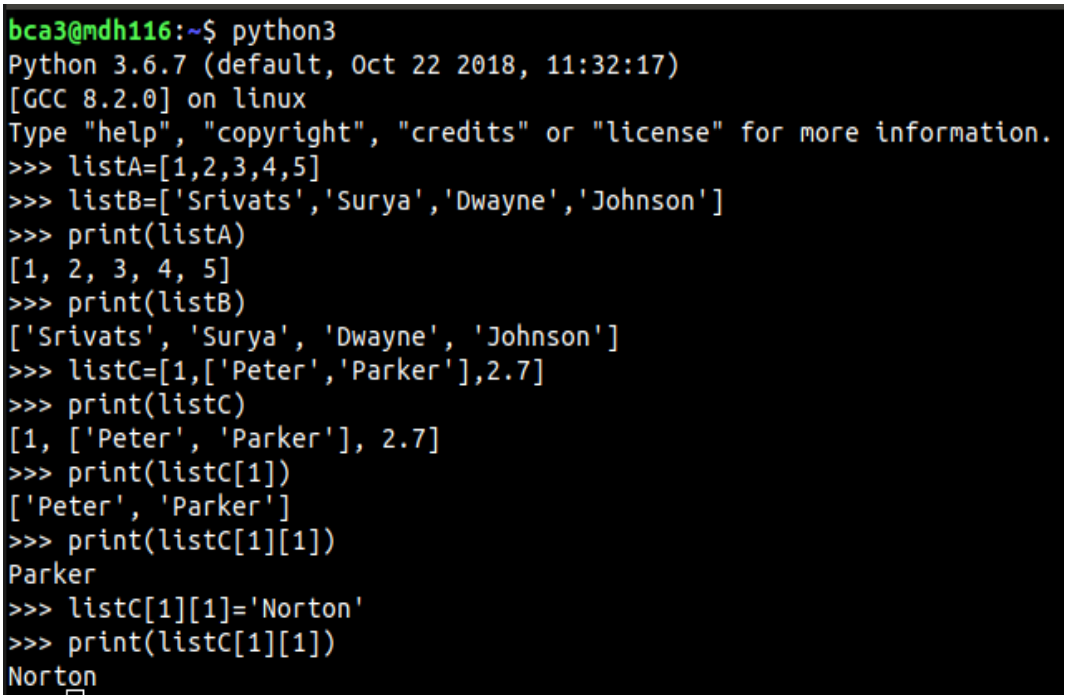
Slowest: Accessing an object field through getattr()

```
$ python3.2 -mtimeit -s"from gender import GenderPronouns; pnoun = GenderPronouns('she', 'her', 'her', 'hers

1000000 loops, best of 3: 0.731 usec per loop
```

4. Grid (Nested List):

A list is a sequence of values. In a string, the values are characters; in a list, they can be any type. The values in list are called elements or sometimes items. A list can be declared as follows:



A list within another list is nested as can be seen in listC in the screenshot. It can be observed that both lists also like dictionaries are mutable i.e. their value can be changed. Hence a list which contain other nested lists and when these nested lists contain values can be called as 2D lists.

5. **Function Calls:**

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

When a function calls another function in order to perform a task is called a **Function Call**. A function call made by one function to itself is called **Recursion**. In this project, function calls and recursion also form an integral part of the core concepts as the moves each piece on the board use multiple function calls in order to analyse and display the move graphically.

Skeletal Framework of the Code:

The Game Development Project has the following skeletal framework along with their respective explanation:

Dictionaries of Pieces:

As it is explained in the core concepts section, dictionaries are used to store the relevant information of each individual piece.

Each individual piece dictionary has the following attributes or rather keys:

1. **Value:**

This attribute is used to store the CLI (Command Line Interface) based board value or denotation. It is used to identify the position of the piece on the CLI board (2D list) which is later interpreted later using the GUI (Graphical User Interface) board.

These values are a series of minimum 2 and maximum 4 characters which denote their colour (Black or White), the piece name(King, Rook ,Pawn, etc.) and its initial position(King-side or Queen-side if it is not a pawn, 1^{st} , 2^{nd} , 3^{rd} , etc. if it is a pawn).

a. **Coords:**

Coords is short for Coordinates. As the name suggests it is used to map the actual

CLI based board coordinates which are later mapped to the GUI coordinates by multiplying them with the TILESIZE global variable which is discussed later. **The coordinates are stored in the form of a list** in the format **[row,coloumn]**.

b. **Image:**

This attribute is used to map to the image which is stored in location 'GameImages/Pieces'. This image is rendered using blit() method by the surface objects in pygame (discussed later) to the screen.

c. **Elimination:**

Elimination is a **Boolean** attribute denotes the removal of a piece from the board. If a piece has been captured by another piece in the game, the captured piece's Elimination is set to **True**. If the Elimination is set to True, the piece will not be printed on the CLI board and hence it will not be rendered on the GUI board either.

d. Move:

This is a pretty interesting attribute. While other attributes store values such as integers, string, Boolean or lists, this attribute stores the function calls for the moves of the respective piece.

Some special Attributes:

Dual-step (in Pawn):

This attribute is a Boolean attribute which is used to detect whether the pawn is eligible to make a move with by jumping 2 boxes. It can only make this move once and as it' s first move only.

Castling (in King):

This attribute is another Boolean attribute which is used to detect the eligibility of Castling for a king. Like Dual-Step, it can only be done once in the game by the King and as it' s first move.

There are 2 sets of dictionaries which are working in the background. Each board piece has 2 sets it' s own values with certain changes. The dictionaries of the pieces are labelled 1 and 2 respectively. These dictionaries are assigned according to the choice of black and white in the game. There is another dictionary called **Pieces** in the game which is used to store the **Value** attribute of different dictionaries as it' s key in order to enable mapping of a particular piece from the CLI board. Just like nested lists, it is like storing a dictionary inside another dictionary.

Even Kings have their own dictionaries in order to enable the smooth detection of Check and Check Mate respectively.

Global Variables (Constants):

The Global variables are the variables which are not declared inside a function. In this project, the global variables are declared in upper case to denote that these values are **constant**. Some of these constants will be discussed later in the report. Apart from them, the other constants are:

BLACK=(0,0,0)
WHITE=(255,255,255)
BLUE=(0,0,255)
GREEN=(0,255,0)
RED=(255,0,0)
TILESIZE=74
BOARDSIZE=592
FPS=100
MAX_BYTES=65535

Pygame Functions:

Pygame module is essentially the core module of this project. It is imported using the statement:

import pygame

But this is not enough. In order to access the functions, variables and classes of the pygame module, the following piece of code must be written:

pygame.init()

The other functions are discussed below according to their usage in the project code:

DS=pygame.display.set_mode((850,592))

UIS=pygame.display.set_mode((850,592))

CH=pygame.display.set_mode((850,592))

PG=pygame.display.set_mode((850,592))

The above function is used to instantiate the surface objects which has the dimensions (850,592). These surface objects can be used to render images on top of it using the blit() method discussed later.

pygame.display.set_caption('Chess')

This piece of code is used to set the title of the window.

FPSCLOCK=pygame.time.Clock()

A Clock object is created in order to run the frame rate which is 100 frames. This clock object runs time in seconds.

LB=pygame.image.load('GameImages/Tiles/LightBrown.png').convert()

DB=pygame.image.load('GameImages/Tiles/DarkBrown.png').convert()

BG=pygame.image.load('GameImages/Backgrounds/Dark-Wood-Background.jpg').convert()

The load() present in the image module imported by pygame module stores the value of the image in the constant. The convert() function converts the image into a background image.

Other pygame functions and methods will be discussed further in the report.

Board Based Functions:

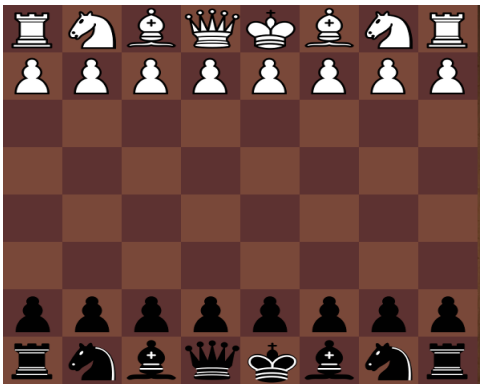
These are the functions which are used to render the playing chess board along with their respective pieces. These functions are categorised as:

CLI Based:

These functions are used to generate the boards in the game for the command line as per the player' s choice. The function used for this is **chessBoard(Choice)** where is the parameter.

The output for **chessBoard(Choice)** function is shown below with comparrision:

```
['11', '00', '11', '00', '11', '00', '11', '00']
['00', '11', '00', '11', '00', '11', '00', '11']
['11', '00', '11', '00', '11', '00', '11', '00']
['00', '11', '00', '11', '00', '11', '00', '11']
['11', '00', '11', '00', '11', '00', '11', '00']
['00', '11', '00', '11', '00', '11', '00', '11']
['11', '00', '11', '00', '11', '00', '11', '00']
['00', '11', '00', '11', '00', '11', '00', '11']
```



Here, '11' stands for black and '00' stands for white. The explanation for the pieces put on the GUI board is the result of another function called **piecesBoard(Pieces)**

The Pieces parameter is the dictionary that contains the keys for other dictionaries i.e. the dictionary of the board pieces.

The output is as follows:

```
['WRQ', 'WKNQ', 'WBQ', 'WQ', 'WK', 'WBK', 'WKNK', 'WRK']
['WP1', 'WP2', 'WP3', 'WP4', 'WP5', 'WP6', 'WP7', 'WP8']
['', '', '', '', '', '', '', '']
['', '', '', '', '', '', '', '']
['', '', '', '', '', '', '', '']
['', '', '', '', '', '', '', '']
['BP1', 'BP2', 'BP3', 'BP4', 'BP5', 'BP6', 'BP7', 'BP8']
['BRQ', 'BKNQ', 'BBQ', 'BQ', 'BK', 'BBK', 'BKNK', 'BRK']
```

GUI Based:

The GUI based functions render the output on the screen using the blit() function. The blit() function takes 2 arguments: Image and a tuple of GUI coordinates.

The GUI coordinates can be obtained multiplying the board based coordinates of each image with the constant TILESIZE which equals 74. Hence, it will render the objects in multiples of 74.

The putTiles(board) function puts the board tiles on the window and putBoardPieces(piBord, Pieces) puts the pieces on the graphical board.

Moves:

There are 3 basic moves which are followed for almost any piece which essentially follow the following basic steps:

a. If in the course of the path to the target location there is a piece in between, then return the move as invalid.

b. If there is a piece at the target location and that piece belongs to the opposite side then set the **Elimination** of the piece in the target location is set to **True** and the selected piece is moved over there. It is also moved if the target location is empty. Else if the piece in the target location belongs to the same side, return the move as illegal.

1. **Horizontal Move:**

   The Horizontal checks the following condition:

   Given a piece, check whether the piece has it' s **initial row position equal to that of the target row position**. If not so, then return the move to be invalid.

2. **Vertical Move:**

   The Vertical move checks the following condition:

   Given a piece, check whether the piece has it' s **initial column position equal to that of the target row position.** If not so, then return the move to be invalid.

3. **Diagonal Move:**

   Given a piece, check whether the piece has the absolute value of the difference between it' s initial row and target row equals to that of the absolute value of the difference between the initial row and the target column. If not so, then return the move to be invalid.

Rook Move: Horizontal Move OR Vertical Move

Queen Move: All the 3 moves

Bishop Move: Diagonal Move

King Move: **All the 3 moves** if the absolute value of the difference between the row and target row, column and target column or both is 1.

Pawn Move: **Horizontal move** if the absolute value of the difference between the row and target row is 1 and **diagonal move** if it has to eliminate a piece of the opposite side along with the condition that the absolute value of the difference between the row and target row and column and target column is 1.

Some special moves:

Knight Move:

The Knight Move is performed using the following logic:

If absolute(row – target row)=1 and absolute(column –target column)=3 or absolute(row – target row)=3 and absolute(column –target column)=1, then return the move as a valid move.

The Knight piece can be placed directly over the target position without worrying about the pieces in between as a Knight can jump over pieces.

Castling:

Castling is the move which the **King** performs when the king has to protect itself. It is done by bringing the Rook closer and putting it before the King while the King moves 2 steps forward (King side Castling) or 3 steps forward (Queen side Castling). The **Castling** is an attribute or key present in the King' s dictionary. **If the King has to**

Castle, then it should be it 's first move and if the Castling attribute of the piece is set to True then, the King cannot Castle.

Empassant Pawn:

The Empassant Pawn move is performed by the pawn in order to eliminate it' s opposite side's piece which is present diagonally while moving 2 steps forward. **It can only be done as the first move.**

Check and Check Mate:

The **Check** and **Check Mate** form the core of the game as they decide the winner of the game. The King is the crucial piece in the course of the entire game. If the King of one side is check mated by that of the opposite side, then the opposite side wins and vice-versa.

The criteria for checking **Check** for the King is as follows:

a. Check if the King is direct line with Rook and Queen of the opposite side horizontally and vertically.
b. Check if the King is in direct contact with the Queen, Bishop and Pawn of the opposite side diagonally. If it is a Pawn, then check whether the absolute values of differences between their rows and columns is 1.
c. Check whether the **King** is in one of the target locations of the **L shaped movement trajectory** of the **Knight**.
d. If either of these conditions is **True**, then it is a **Check**.

The criteria for checking a **Check Mate** for the King is as follows:

a. Check if the King is Checked by any other piece on the board.
b. If it is a Check, the Check for the following criteria:
    a. **Piece Attacker**:

    Checking whether the piece that Checked the King is being attacked by another piece. We simply do that by applying the Check function on that particular piece.

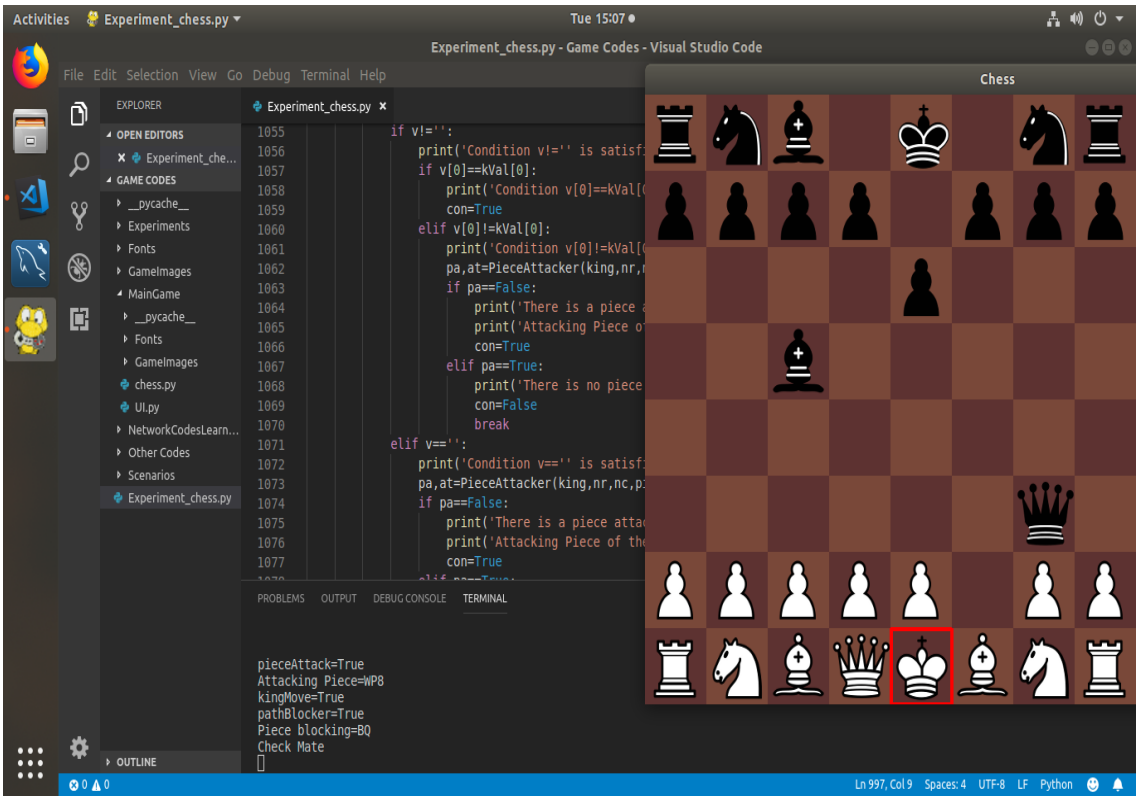
b. **King Movement**:

Checking whether the King cannot move to any other location without being checked by another piece in that position.

c. **Path Blocker**:

Checking whether the path between the King and the Checking piece can be blocked by a piece that belongs to the playing side of the Checked King.

If all of these conditions are True, then it is a **CHECK MATE**.



Stalemate:

Stalemate is the situation in which one side except for the King faces elimination of all the pieces. During this, the opposite side has **50 moves in order to Check Mate the King**. **If it does succeed in doing so then the game is a draw.**

Network Programming:

Network Programming deals with writing piece of code in order to establish connection between 2 machines over a network. It is done by importing the socket module in the code. In order to begin any networking program or applying an Internet Protocol (IP), a socket has to be created. This project as mentioned in the previous sections uses **UDP (User Datagram Protocol)** in order to establish the connection between 2 systems. Creating a UDP socket is done simply by writing the following piece of code:

my_socket=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)

Here, we are creating a socket object. It belongs to the **AF_INET family of internet protocols** and **SOCK_DGRAM** mentions that it is a **UDP socket.**

Then, if the socket is created to the **server**, bind() method is used in order bind the IP address and the port number in order to enable it to receive datagram from any client.

The following piece of code shows the socket for a server.

my_sock.bind((IPaddr,Port))

For a **client**, connect() method is used in order to establish connection with the server.

my_sock.connect((IPaddr,Port))

Data is sent to the other side by encoding the message in **ASCII**. It has to also be decoded in ASCII for viewing. The code is as follows:

Sending:	**Receiving:**
Message='Hello World'	***data,address=my_sock.recvfrom(MAX_BYTES)***
Data=Message.encode('ascii')	***message=data.decode('ascii')***
my_sock.send(MAX_BYTES)	***print(str(message))***

Here, **MAX_BYTES** is the buffer size which is passed as a parameter while sending and receiving. In this project **MAX_BYTES equals 65535 bytes**.

In this project, the **Host()** function acts as the server. The **Guest()** function acts as the client. More details are discussed in the application workflow part of the report.

The game() function:

This function controls the entire gameplay of the game by:

a. Listening for events in the game
b. Selecting the pieces and moving them
c. Testing for Check, Check Mate and Stalemate
d. Rendering the required graphics on the window surface.
e. Finishing the entire game as a result of the multiple iterations

The events in the game are given as mouse inputs. It check the events using the following Python pseudo code:

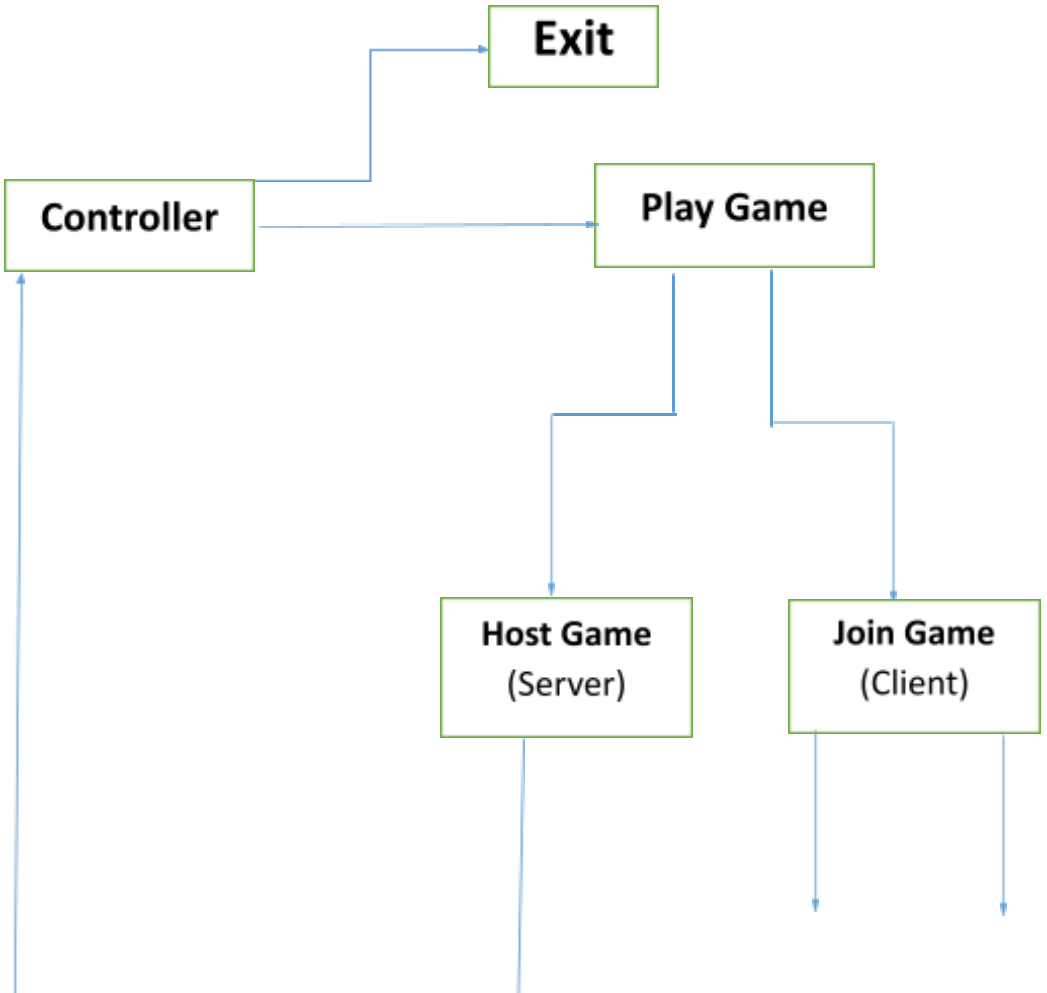
```
for event in pygame.event.get():
    if event.type==QUIT:
        pygame.quit()
        sys.exit()
    elif event.type==MOUSEMOTION:
        #Do something
    elif event.type==MOUSEBUTTONUP:
        mousex,mousey=event.pos
        #Use it to select a piece on the board
```

The **chance** variable is a Boolean value which determines the turn of the players on the board. The function boardCoordinate(mousex,mousey) map the particular piece which has been selected on the GUI board. After the piece has been selected, the piece is moved from the value stored in the dictionary' s move attribute. If the move is True (valid) then the chance negated to change the turn in the gameplay.

Check and Check Mate is tested every time a move is made along with the graphical rendering of the board pieces on the board.

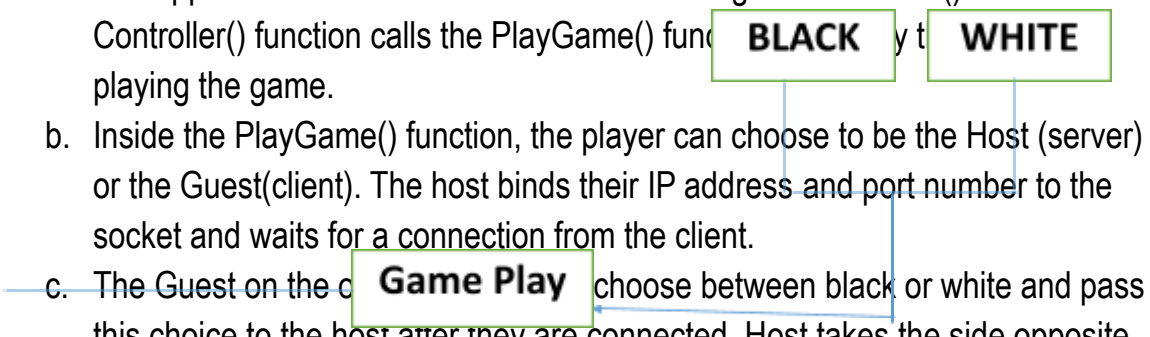
The **CutPieces()** function takes care of all the eliminated pieces by putting them on the side of the board as per their playing side.

Application Workflow:



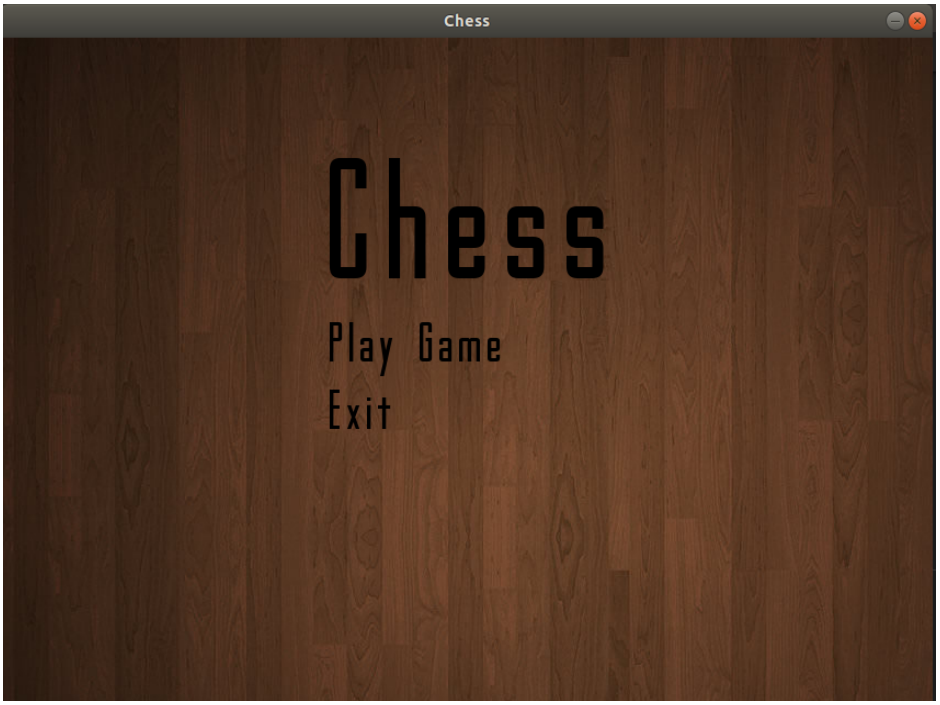
The Application works as follows:

a. The application renders it' s User Interface using the Controller() function. The Controller() function calls the PlayGame() fun y t playing the game.
b. Inside the PlayGame() function, the player can choose to be the Host (server) or the Guest(client). The host binds their IP address and port number to the socket and waits for a connection from the client.
c. The Guest on the c choose between black or white and pass this choice to the host after they are connected. Host takes the side opposite to the side that is chosen by the Guest. Both the Host and the Guest play the game.
d. After a winner is decided by the algorithm of the gameplay (game() function), the Host and the Guest disconnect.
e. The Controller() function will be executed again using recursion until the **Exit** option is clicked by the player.

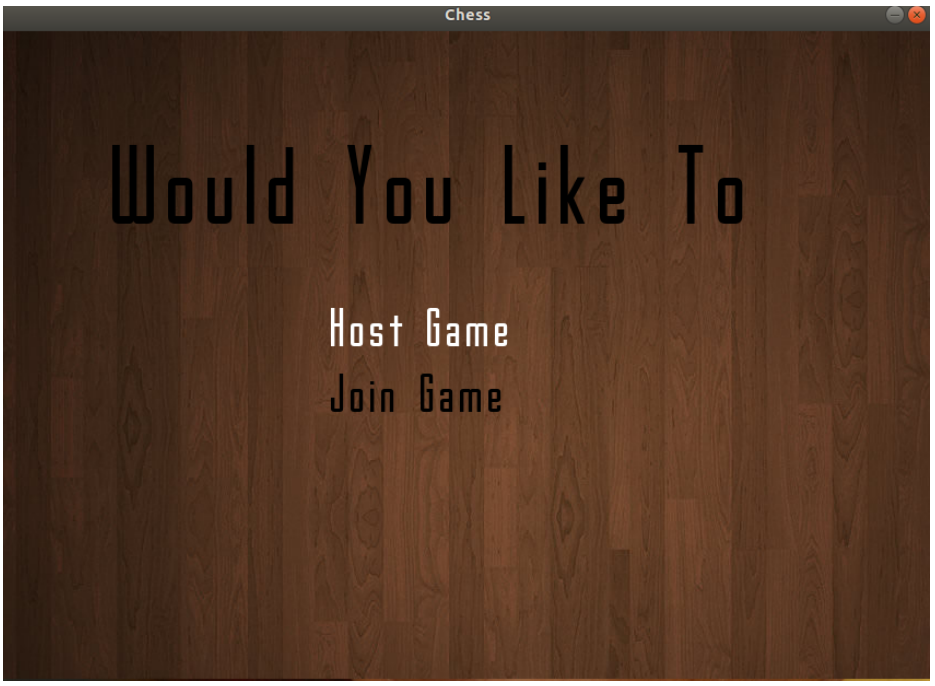


Screenshots:

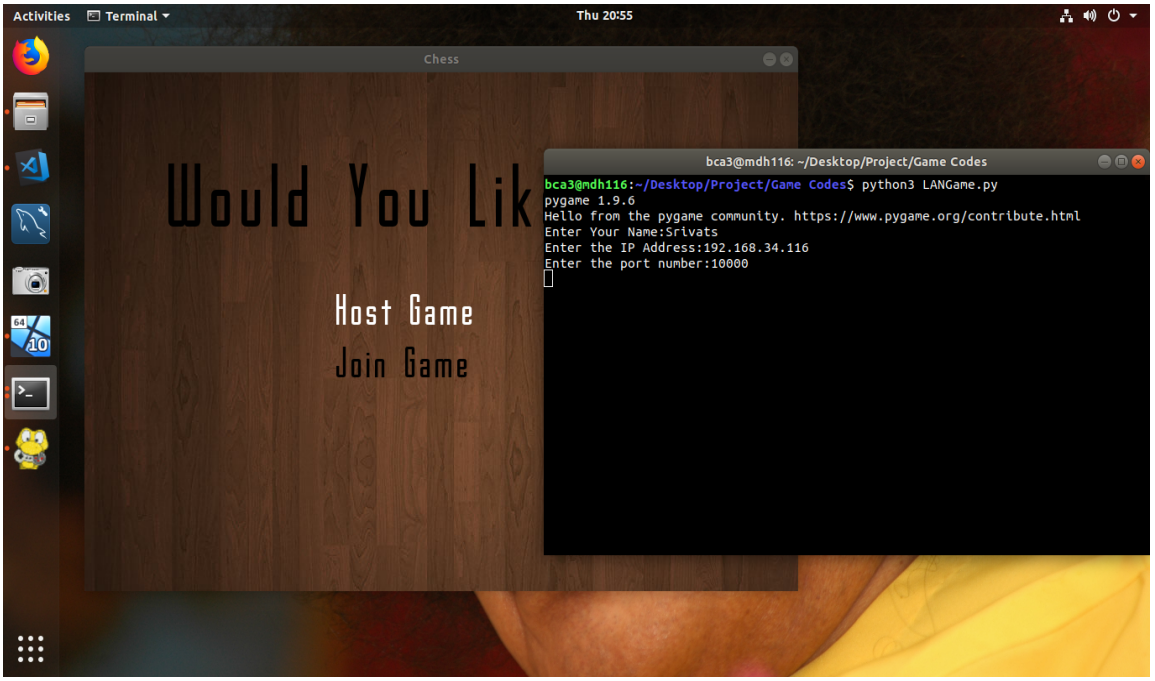
1. **Main User Interface:**



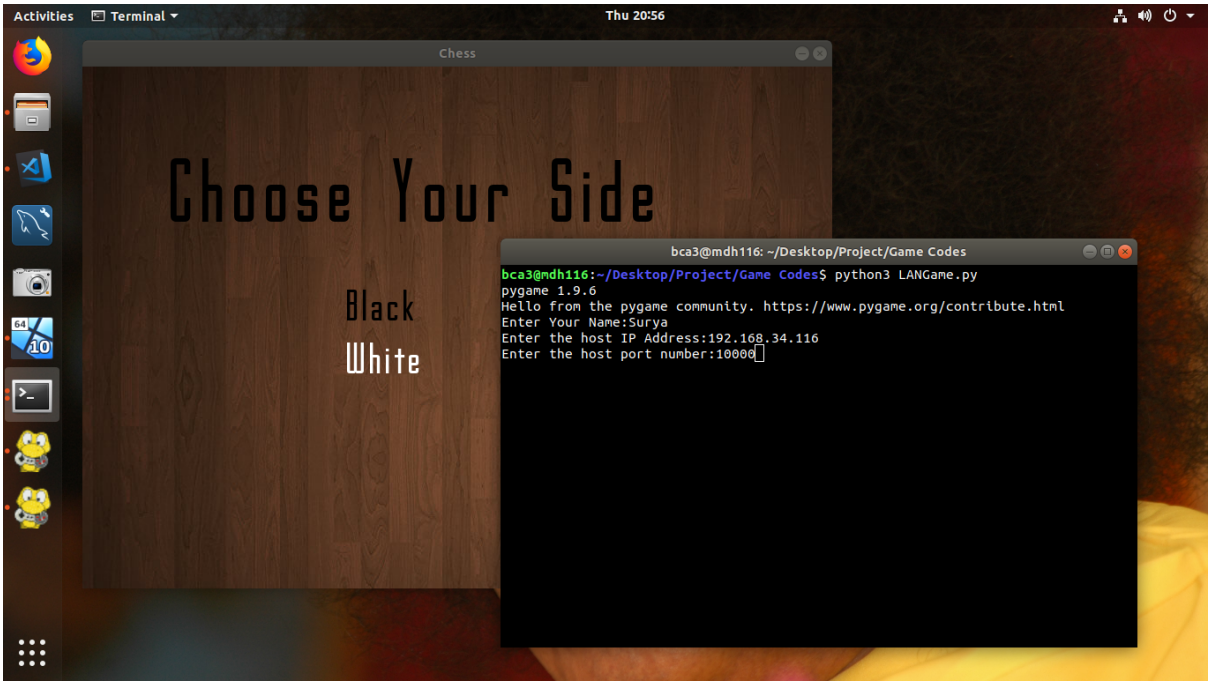
2. **Play Game Interface:**



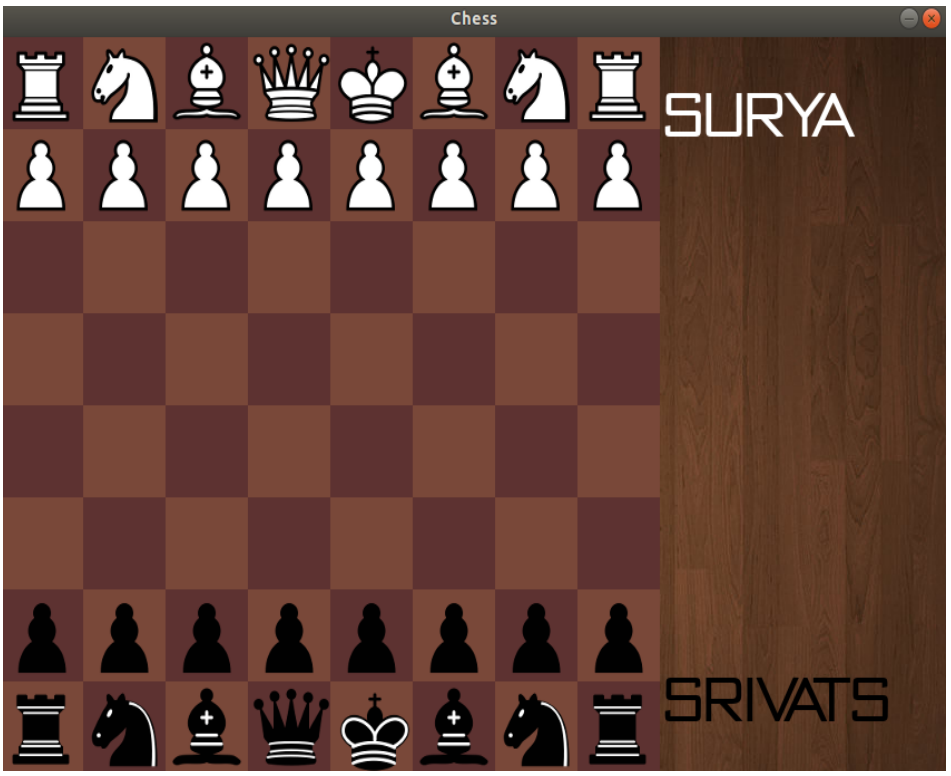
3. Host Game Terminal:



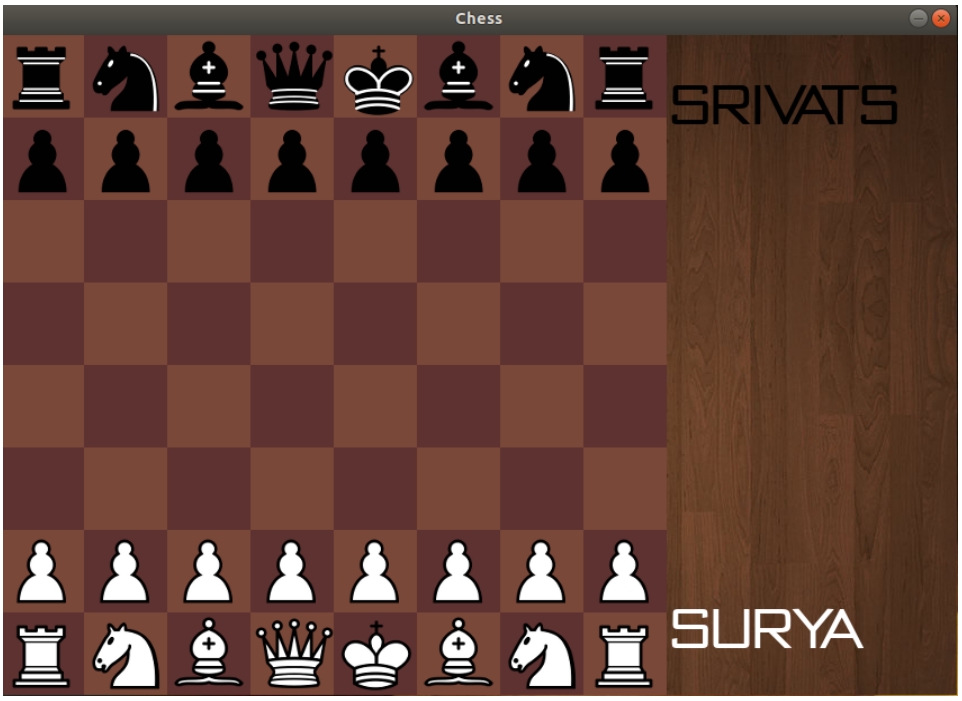
4. Join Game With Terminal:



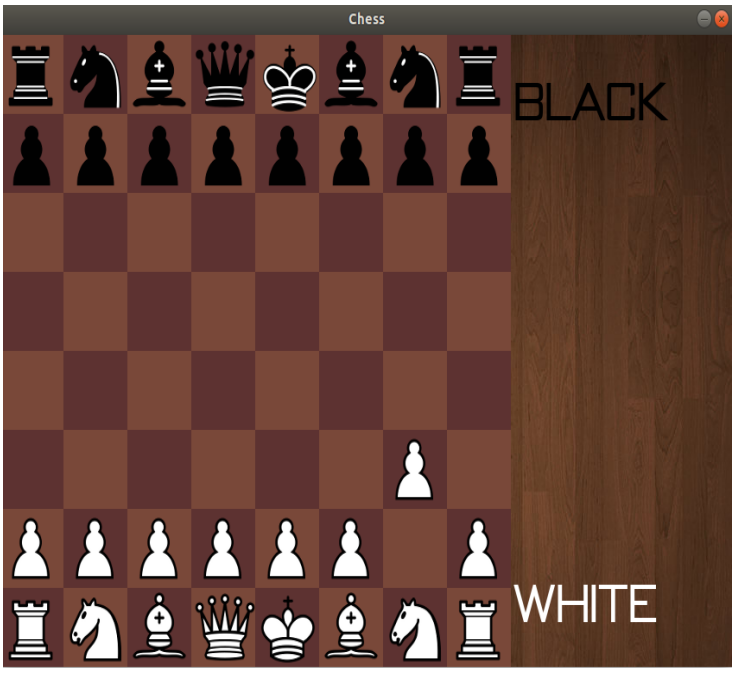
5. Host to Guest View:



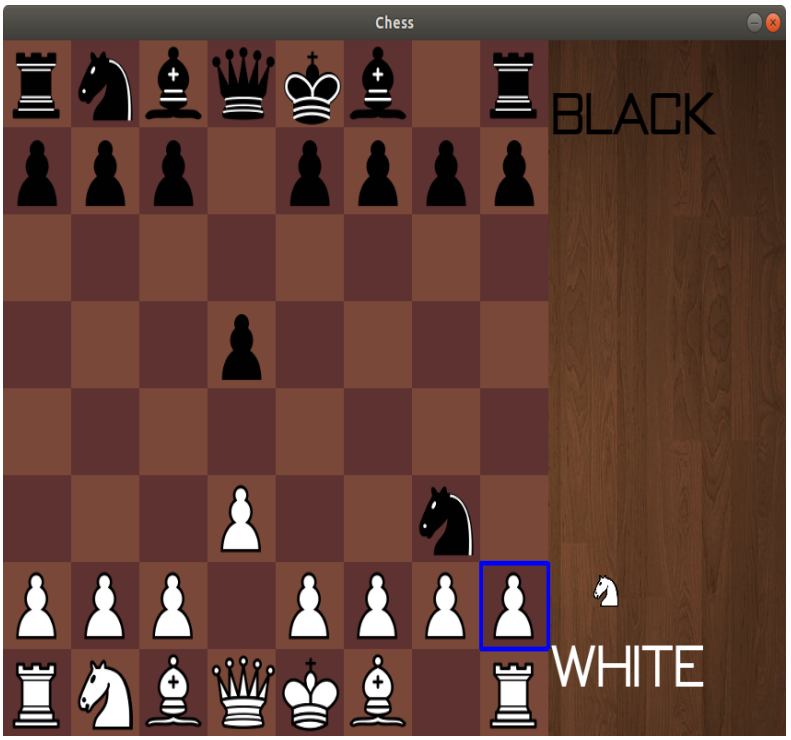
6. **Guest to Host View:**



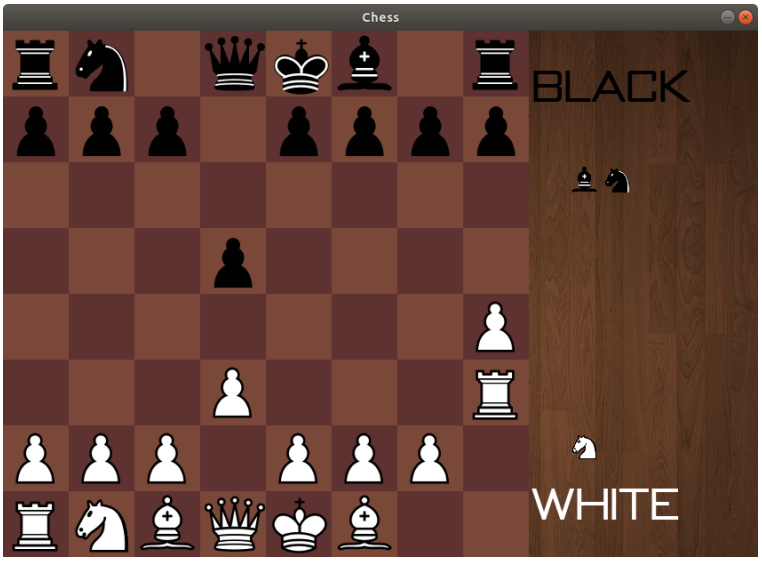
7. Pawn Move:



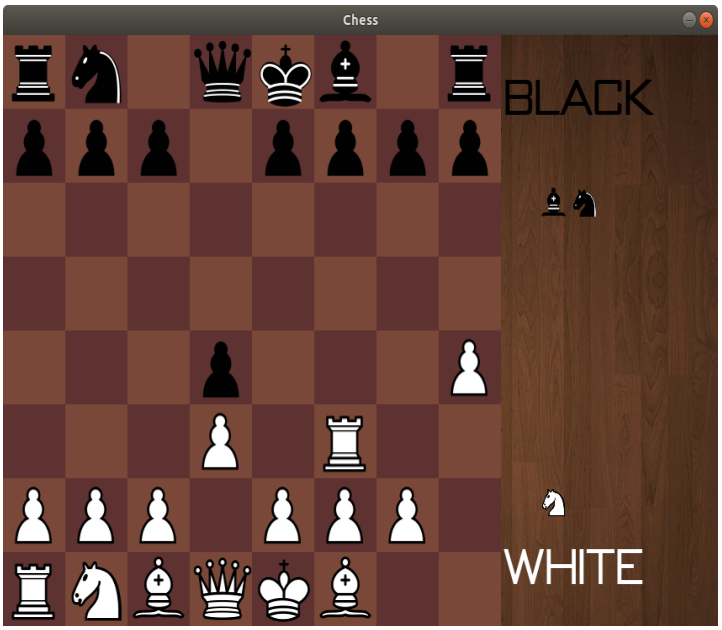
a. Empassant Pawn:



8. **Rook Move:**
    a. **Vertical Move:**

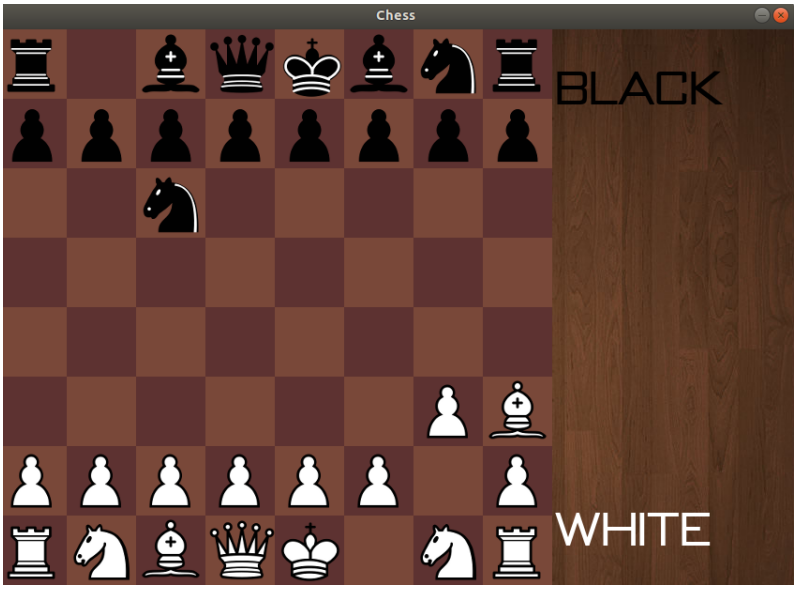


b. **Horizontal Move:**



9. **Knight Move:**

10. Bishop Move:



11. Queen Move:

a. Vertical:

b. **Horizontal:**

c. **Diagonal:**

12. **King Move:**
    a. **Vertical Move:**

    b. **Horizontal Move:**

c. **Diagonal Move:**

d. King Castling:

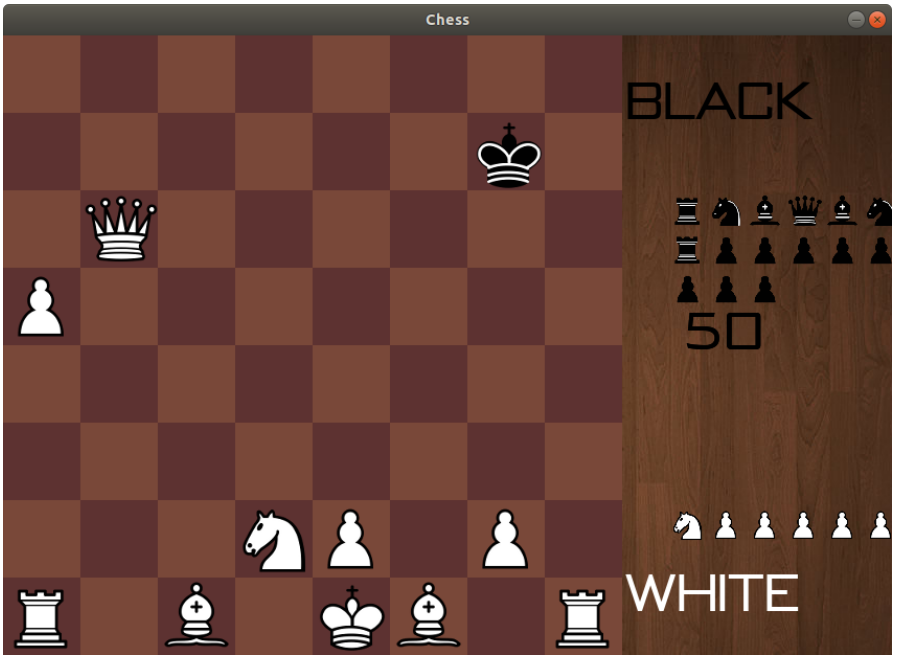
13. Check:

14. 4 move Check Mate:

15. **Stale Mate:**

a. **In Progress:**

b. **Complete:**



Future Scope

This game has been developed as means of bringing about an inception of e-sports in the campus. If the bugs regarding networking are fixed and other moves like Pawn Promotion are added, then it can be completed as a full functioning game. Later database support can be added in order to make this game a full-fledged application that can keep track of player's information such as rank, matches won, skill tier, etc. Move timers can be added and points of the players can be kept track of by the application.

Overall, this game according to me should become an e-sports game in this campus if the above requirements are added.

Conclusion:

This project was aimed to bring about e-sports in this campus. Although this project was not successful, this project stills holds the potential to become a successful game if more features are added and certain moves are edited. This project was an attempt to run an optimised developer friendly version of the chess by using Python. It was a learning experience as when one builds this project, they get to learn about game development and Python language along with certain parts of networking.

Overall this project can be termed as partially completed project.

Bibliography:

http://www.stackoverflow.com

http://www.github.com

http://www.realpython.com

http://www.geeksforgeeks.com

http://www.pythonwiki.com

Making Games with Python and Pygame - ***Albert Sweigart***

Foundation of Python Network Programming –*Brandon Rhodes and John Goerzen*