NEW HORIZON COLLEGE OF ENGINEERING

Autonomous College Permanently Affiliated to VTU, Approved by AICTE & UGC
Accredited by NAAC with 'A' Grade, Accredited by NBA

A MINI PROJECT REPORT

for

Mini Project using Python (20CSE59)

on

CHESS - CLASH OF KINGS MULTIPLAYER

Submitted by

J Karthik Surya

USN: 1NH19CS236, Sem-Sec: 5-D

In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

Academic Year: 2021-22

NEW HORIZON
COLLEGE OF ENGINEERING
Autonomous College Permanently Affiliated to VTU, Approved by AICTE & UGC
Accredited by NAAC with 'A' Grade, Accredited by NBA

CERTIFICATE

This is to certify that the mini project work titled

CHESS - CLASH OF KINGS MULTIPLAYER

submitted in partial fulfillment of the degree of Bachelor of Engineering in Computer Science and Engineering by

J Karthik Surya

USN:1NH19CS236

DURING

ODD SEMESTER 2021-2022

for

Course: Mini Project using Python-19CSE48

Signature of Reviewer

Signature of HOD

SEMESTER END EXAMINATION

Name of the Examiner *Signature with date*

1.__ _____

2.__ _____

1NH19CS236

ORIGINALITY REPORT

3%	%	3%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Mayergoyz, Isaak. "Nonlinear Impedance Boundary Conditions and Their Application To The Solution of Eddy Current Problems", Nonlinear Diffusion of Electromagnetic Fields, 1998. Publication	1%
2	G.Sophia Jasmine, D.Magdalin Marry, S.Swetha Lakshmi, R. Rishiwanth, K. Sreehariprasath, J. Surendhar. "Camera based text and Product Lable Reading for Blind People", 2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS), 2021 Publication	1%
3	B. N. Madhukar, Sanjay Jain. "A duality theorem for the discrete sine transform (DST)", 2015 International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT), 2015 Publication	1%

4 Lucas Nunes Faria. "Análise bimodal de teletransporte de estados quânticos da luz no domínio da frequência", Universidade de Sao Paulo, Agencia USP de Gestao da Informacao Academica (AGUIA), 2020 <1%
Publication

5 Marta Chromá. "Synonymy and Polysemy in Legal Terminology and Their Applications to Bilingual and Bijural Translation", Research in Language, 2011 <1%
Publication

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	On		

ABSTRACT

The primary objective in chess is to checkmate your opponent's King. When a King cannot avoid capture then it is checkmated and the game is immediately over. If a King is threatened with capture, but has a means to escape, then it is said to be in check.

A King cannot move into check, and if in check must move out of check immediately. There are three ways you may move out of check: Capture the checking piece; Block the line of attack by placing one of your own pieces between the checking piece and the King. (Of course, a Knight cannot be blocked.); Move the King away from check.

If a King is not in check, and no other legal move is possible, then the position is said to be in stalemate. A stalemated game is a draw, or a tie.

To provide a user-friendly interactive environment to the users of the application that helps them to play and communicate with a lot of ease. To provide help to the users in playing the chess that is the different moves of the different pieces etc are being explained to the users, if they require.

Since there exists client and server as the project is based on client server architecture, where server is serving as a mediator in between the players and the client is making request to server as well as doing all the part that is related to playing logic.

The care has been taken that the application has less CPU usage, so that other applications can also be performed, if required.

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be impossible without the mention of the people who made it possible, whose constant guidance and encouragement crowned our efforts with success.

I have great pleasure in expressing gratitude to **Dr. Mohan Manghnani**, Chairman, New Horizon Educational Institutions, for providing necessary infrastructure and creating good environment.

I take this opportunity to express my profound gratitude to **Dr. Manjunatha,** Principal, New Horizon College of Engineering, for his constant support and encouragement.

I would like to thank **Dr. Amarjeet Singh**, Professor and Dean-Academics, NHCE, for his valuable guidance.

I would also like to thank **Dr. B. Rajalakshmi**, Professor and HOD, Department of Computer Science and Engineering, for her constant support.

I also express my gratitude to Dr./ Ms./ Mr. **Faculty Name**, Designation, Department of Computer Science and Engineering, my mini project reviewer, for constantly monitoring the development of the project and setting up precise deadlines. Her / His valuable suggestions were the motivating factors in completing the work.

J Karthik Surya

USN: 1NH19CS236

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 PROBLEM DEFINITION

The primary objective in chess is to checkmate your opponent's King. When a King cannot avoid capture then it is checkmated and the game is immediately over. If a King is threatened with capture, but has a means to escape, then it is said to be in check. A King cannot move into check, and if in check must move out of check immediately. There are three ways you may move out of check: Capture the checking piece; Block the line of attack by placing one of your own pieces between the checking piece and the King. (Of course, a Knight cannot be blocked.); Move the King away from check. If a King is not in check, and no other legal move is possible, then the position is said to be in stalemate. A stalemated game is a draw, or a tie.

1.2 OBJECTIVES

- **Objective of material**: Players with a lot of material (coins) can use additional power to overwhelm their opponents, so get the material if possible.
- **Objective of Development**: A well-developed character has more firepower than an underdeveloped character, so maximize your character's development.
- **Objective of Centre-control**: Most actions take place within or through the centre, so try controlling the centre.
- **Objective of King-safety**: Unmasked kings are very vulnerable, so protect them and expose your opponent's kings.
- **Objective of Pawn-structure**: Pawn structures affect the development of pieces and determine where weak squares are, so keep your pawn structure strong and weaken your opponent's pawn structure.

1.3 EXPECTED OUTCOMES

To provide a user-friendly interactive environment to the users of the application that helps them to play and communicate with a lot of ease. To provide help to the users in playing the chess that is the different moves of the different pieces etc are being explained to the users, if they require. Since there exists client and server as the project is based on client server architecture, where server is serving as a mediator in between the players and the client is making request to server as well as doing all the part that is related to playing logic. The care has been taken that the application has less CPU usage, so that other applications can also be performed, if required.

1.4 REQUIREMENTS SPECIFICATION

Operating system : Windows

Programming Language : Python.

IDE/Workbench : Pycharm

Processor : Pentium IV or higher

Hard Disk : 40GB

RAM : 256MB or above

CHAPTER 2

FUNDAMENTALS OF PYTHON

2.1 INTRODUCTION TO PYTHON

Python is a highly interpreted general-purpose programming language. This design philosophy emphasizes the readability of the code with important indentation. Its linguistic structure and object-oriented approach are designed to allow programmers to write clear and logical code for large and small projects. Python is dynamically typed and garbage is collected. It supports several programming paradigms, including structured (especially procedural), object-oriented, and functional programming. Due to its extensive standard library, it is often referred to as the "battery-included" language. Guido van Rossum began developing Python as a successor to the ABC programming language in the late 1980s and first published it in 1991 as Python 0.9.0. Python 2.0 was released in 2000 and introduced new features such as list comprehensions and a circular garbage collection detection system "in addition to counting references".

Python 3.0 was released in 2008 and was a major overhaul of languages that were not fully backward compatible. Python 2 was deprecated in version 2.7.18 in 2020.

Figure 2.1: Features of Python

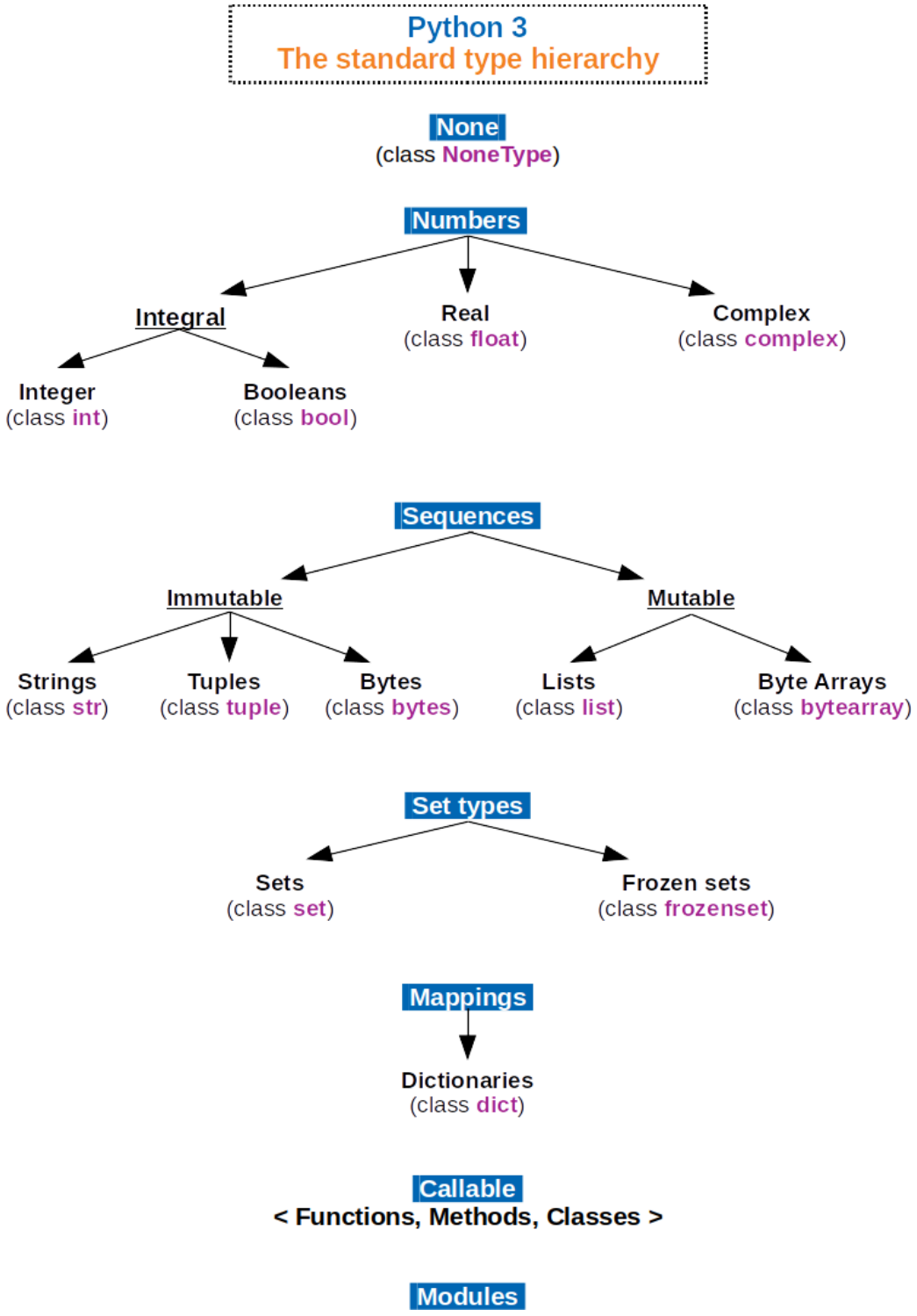

Figure 2.2: Python standard types in hierarchal order

2.2 LISTS

Lists are similar to dynamically sized arrays declared in other languages (Vector in C++ and ArrayList in Java). The list doesn't always have to be uniform, making it the most powerful tool in Python. A list can contain data types such as integers, strings, and objects. Lists are mutable, so you can change them after they've been created.

Lists in Python are ordered and numbered. The elements of the list are indexed in a specific order, and the list is indexed first, starting at index 0. Each element in the list occupies a specific position in the list, so duplicate elements in the list are allowed, and each element has its own value. place and reliability.

2.3 SLICING

Slicing a list in Python is a common practice and is the most commonly used technique by programmers for efficient problem solving. Considering the Python list Inorder, accessing a range of items in the list requires fragmenting the list. One way to do this is to use the simple slice operator: the colon (:).

This operator allows you to specify where the slice starts, ends, and in steps. Splitting a list returns a new list from the old one.

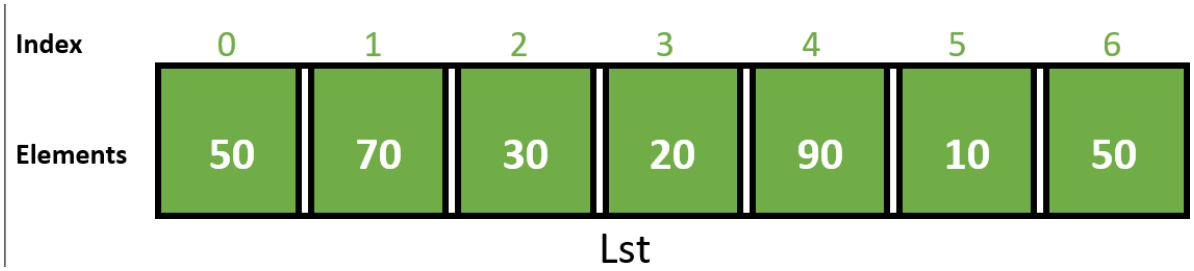

Figure 2.3: Indexing a list using slicing

2.4 BRANCHING

When an "algorithm" chooses one of two (or more) operations, it is called a branch. The most common software "statement" used for navigation is the "IF" statement.

Algorithms in Computer Programs are a lot like recipes, but most recipes don`t allow the cook to make choices. If it says, mix two eggs with a cup of flour, that is exactly what you do. But some recipes do allow for variations. Such as, if cooking on the Grill outside, do one thing, if cooking in the oven inside do something else.

In a computer program, the algorithm often must choose to do one of two things depending on the "state" of the program. If the grade is greater than 90, then give the student an A, otherwise if the grade is greater than 80, give the student a B,... etc.

The else statement can be combined with an if statement. The else statement contains a block of code that is executed if the conditional expression of the if statement evaluates to 0 or FALSE. The else statement is optional and there can be only one else statement after the if.

The elif statement allows you to test multiple expressions for TRUE and execute a block of code as soon as one of the conditions evaluates to TRUE. As with anything else, the elif clause is optional. However, unlike else, which can have at most one statement, there can be any number of elif statements after if.

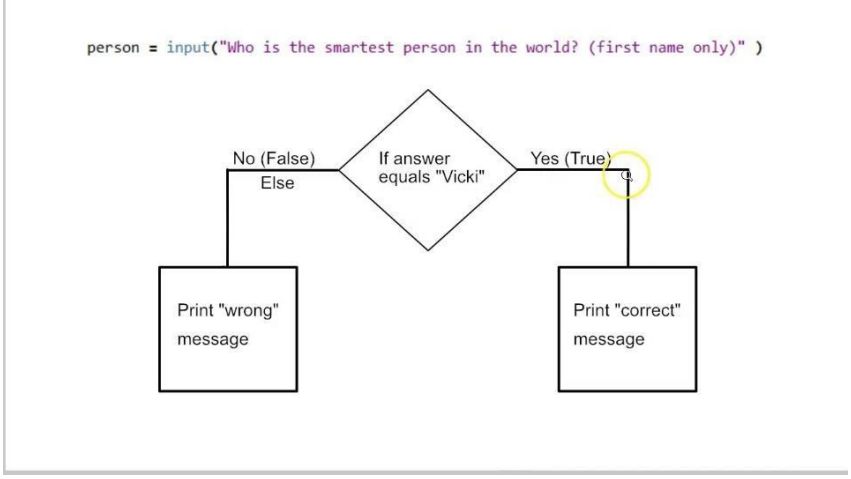

Fig 2.4 BRANCHING

2.5 LOOPS

The Python programming language provides the following types of loops to handle your loop requirements. Python provides three ways to do a loop. All methods provide similar basic functionality, but differ in syntax and when conditions are checked.

While Loop: Python uses a while loop to repeatedly execute a block of statements until a specified condition is met. And when the condition becomes false, the line immediately after the loop in the program is executed.

Nested Loops: The Python programming language allows one loop to be used within another. The following sections show some examples to illustrate the concept.

for in loop: For loop is used for sequential traversal. For example: iterate over a list, string or array, etc. There is no C-style for a for loop, i.e. for (i = 0; i & n; i++) in Python. There is a for in loop similar to the for loop in other languages. Let's see how to traverse sequentially using a for loop.

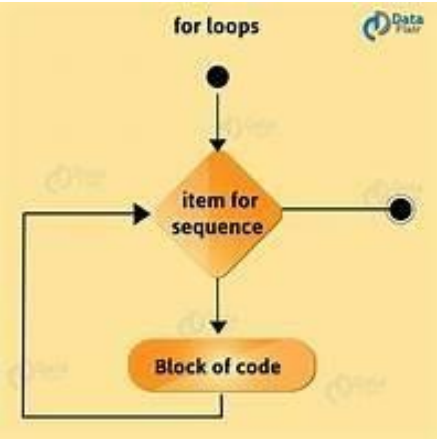

FIG 2.5 LOOPS

2.6 CLASS

A class is a custom blueprint or prototype from which objects are created. Classes provide a means to bring data and functionality together. Creating a new class creates an object of a new type, allowing you to create new instances of that type. Properties can be attached to each instance of a class to maintain state. Instances of a class may also have methods (defined by the class) that change state. To understand the need to create a class, let's say you want to keep track of the number of dogs that can have different attributes, such as breed, age, for example. If you are using a list, the first factor could be the breed of the dog and the second factor could be the age. This class instantiates this class to create a user-defined data structure that contains its own data members and member functions that you can access and use. A class is like a project of an object.

2.7 FUNCTIONS

Python Functions is a block of associated statements designed to carry out a computational, logical, or evaluative task. The concept is to position a few generally or time and again carried out responsibilities collectively and make a characteristic in order that in place of writing the equal code time and again for extraordinary inputs, we are able to do the characteristic calls to reuse code contained in it time and again

You can define features to provide the functionality you need. Here are some simple rules for defining functions in Python:

- A function block begins with the keyword def, followed by the function name and parentheses (()).
- All input parameters or arguments must be enclosed in these parentheses. You can also define parameters within these square brackets.
- The first statement in a function can be an optional statement (function docstring or docstring). The code block for each function starts with a colon (:) and is indented.
- The return [expression] statement exits the function and optionally passes an expression to the caller. A return statement with no arguments is equivalent to return None.

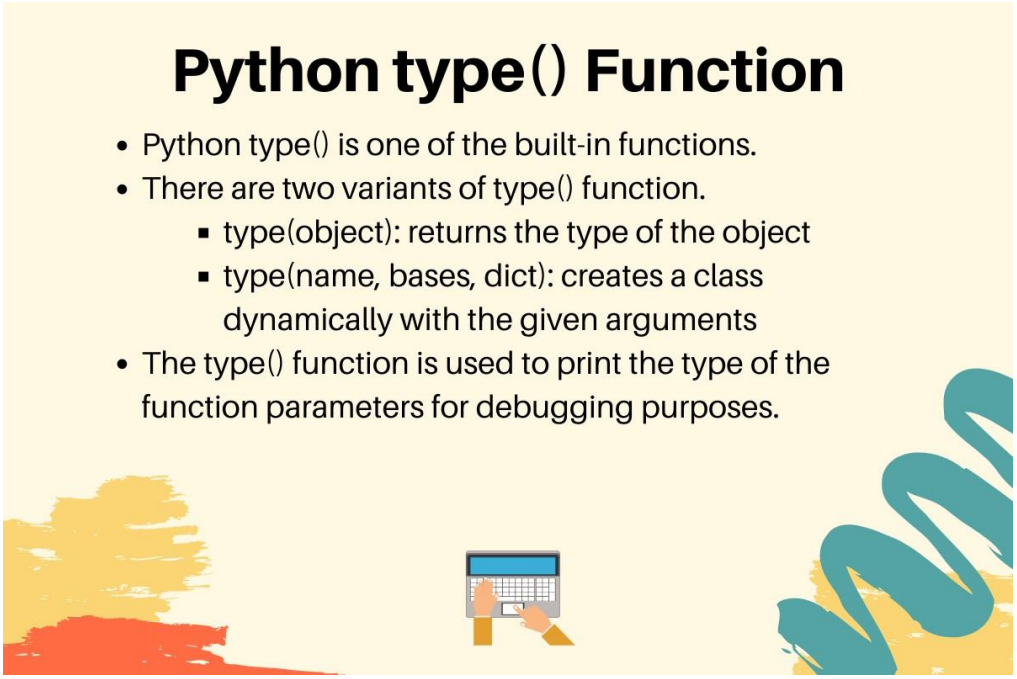

Fig 2.6 PYTHON FUNCTION

CHAPTER 3

FUNDAMENTALS OF PYQT5

3.1 INTRODUCTION

PyQt is a Python binding for the open-source Qt widget toolkit that also works as a cross-platform application development platform. Qt is a popular C++ framework for writing GUI applications for all major desktop, mobile and embedded platforms (Linux, Windows, MacOS, Android, iOS, Raspberry Pi, etc. supported).

PyQt is free software developed and maintained by Riverbank Computing based in the UK, and Qt is being developed by the Finnish company The Qt Company.

3.2 WIDGETS

- QLabel is one of the simplest PyQt5 widgets that can display lines of text. This widget has many helper functions and methods that allow you to retrieve, update, and format this text at any time.
- QPushButton is one of the simplest and most common widgets in PyQt5. As the name suggests, a button that triggers a function when clicked (when pressed).
- QRadioButton Radio buttons are typically found in graphical user interfaces where the user is presented with a list of options. Unlike CheckBoxes, you can select only one radio button from multiple (groups).
- Qcheckboxes are an important part of the GUI used to present a list of options to the user. PyQt5 has a widget called QCheckBox that I use to create a checkbox or checkbox button when it is called.
- QInputDialog, PyQt5 has a QInputDialog widget that can create different input dialogs to accept input in different ways.

3.3 Features of PyQT5

Here are important features of PyQt5 which consists of more than six hundred classes covering a range of features such as

- Graphical User Interfaces
- SQL Databases
- Web toolkits
- XML processing
- Networking

3.4 PyQt5 Directory Structure

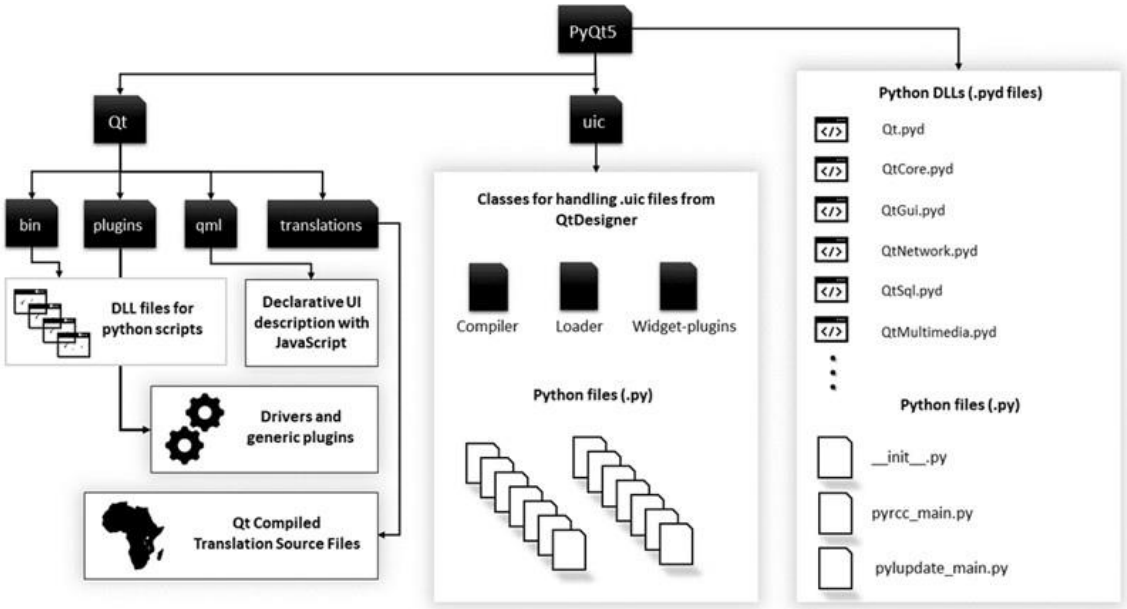

Fig 3.1

3.5 FUNDAMENTAL MODULES IN PYQT5

Qt: Combine all classes/modules mentioned below into one module. This will significantly increase the memory used by the application. However, it is easier to manage the framework by importing only one module.

QtCore: Contains basic non-graphics classes used by other modules. This is where the Qt event loop, signals, slot hooking, etc. avatar.

QtWidgets: Contains most of the widgets available in Pyqt5.

QtGui: Contains GUI components and extends QtCore modules.

QtNetwork: Contains classes used to implement network programming via Qt. Supports TCP server, TCP socket, UDP socket, SSL handling, network session and DNS lookup.

QtMultimedia : It provides low-level multimedia features.

QtSql: Implement database integration for SQL databases. Supports ODBC, MySQL, Oracle, SQLite and PostgreSQL.

CHAPTER 4

DESIGN GOALS

To provide a user-friendly interactive environment to the users of the application that helps them to play and communicate with a lot of ease. To provide help to the users in playing the chess that is the different moves of the different pieces etc are being explained to the users, if they require. Since there exists client and server as the project is based on client server architecture, where server is serving as a mediator in between the players and the client is making request to server as well as doing all the part that is related to playing logic. The care has been taken that the application has less CPU usage, so that other applications can also be performed, if required.

Widget	What it does
QCheckbox	A checkbox
QComboBox	A dropdown list box
QDateEdit	For editing dates and datetimes
QDateTimeEdit	For editing dates and datetimes
QDial	Rotatable dial
QDoubleSpinbox	A number spinner for floats
QFontComboBox	A list of fonts
QLCDNumber	A quite ugly LCD display
QLabel	Just a label, not interactive
QLineEdit	Enter a line of text
QProgressBar	A progress bar
QPushButton	A button
QRadioButton	A toggle set, with only one active item
QSlider	A slider
QSpinBox	An integer spinner
QTimeEdit	For editing times

TABLE 4.1: Various widgets available in PYQT5

4.1 ALGORITHM

- Draw grid draws the borders of the grid (hence the black horizontal and vertical lines separating the tiles), draw grid and make grid create a 2d list that we will use later to access all the nodes.

- Then we create a window object that appears when the chess game starts. I set the size to 800x800, which is the argument of the tuple I passed. Later you can resize the board according to the image of the chess pieces.

- Next, we assign the moves for every chess piece and the obstruction moves too so that that particular chess piece can move according to the provided condition.

- The main thing that makes this program work are these node objects, they are just containers in which chess pieces are stored (in this case they are modelled as tiles on a chessboard. Rows, columns, x, y coordinates.

- I need them separately because all these nodes are stored in an 8x8 2d list. So if you call it you need to call it with row and column numbers whereas you need x to draw to the screen. , y pixel values (you could do a row * 100, but you could add this number 1 to your entire code and forget to see how everything breaks).

- A move function is used to move a piece on the screen (eg a black and white pattern), and an adjustment method is used to move an image to the screen if there is a piece in that position on the screen.

- I need to find out if the user has clicked on a tile, which is what find node does when the user taps the screen. show_possible_moves is a feature which makes it stand out by getting a list of possible moves and changing the tile colour for these moves.

- Check and checkmate conditions are the used, when the king is checked, that particular king's tile is displayed in red colour, and when it is checkmated the game is over!

CHAPTER 5

IMPLEMENTATION

5.1 Main code with chess algorithms

```
grid=[(['.']*8)[:] for x in range(8)]
print(grid)

#W- white player B-black player

#turn=1
#K-king, Q-queen, H-knight, B-bishop, P-pawn, R-rook

def print_grid():
    global grid
    for x in grid:
        print(x)

def direction(x,y,ex,ey):
    val1,val2=-1,-1
    if ex-x>0:
        val1=1
    elif ex-x==0:
        val1=0

    if ey-y>0:
        val2=1
    elif ey-y==0:
        val2=0

    return val1,val2

def obstructed(x,y,ex,ey):
    global grid
```

Fig 5.1

```
def obstructed(x,y,ex,ey):
    global grid
    dirx,diry=direction(x,y,ex,ey)
    stx,sty=x+dirx,y+diry
    while(stx!=ex or sty!=ey):
        if grid[stx][sty]!='.':
            return False
        stx+=dirx
        sty+=diry
    return True

def get_last_coin(x, y, dx, dy):
    pass


def check(bx=-1,by=-1,wx=-1,wy=-1,mode='D'):
    global grid
    Wcheck, Bcheck = False, False
    # bx, by, wx, wy = 0, 0, 0, 0
    if (bx,by,wx,wy)==(-1,-1,-1,-1):
        for x in range(8):
            for y in range(8):
                if grid[x][y] == 'BK':
                    bx, by = x, y
                if grid[x][y] == 'WK':
                    wx, wy = x, y
```

Fig 5.2

```
    if wx>=0:
        for x in range(8):
            for y in range(8):
                if grid[x][y] != '.':
                    if movable(grid[x][y],x,y,bx,by) and grid[x][y][0]=='W':
                        Bcheck=True

    if bx>=0:
        for x in range(8):
            for y in range(8):
                if grid[x][y]!='.':
                    if movable(grid[x][y], x, y, wx, wy) and grid[x][y][0]=='B':
                        Wcheck = True
```

Fig 5.1

```
    print("Check Function Was Called!!")
    if mode=='D':
        return ((wx,wy),(bx,by),Wcheck,Bcheck)
    elif mode=='W':
        return Wcheck
    elif mode=='B':
        return Bcheck

def check_mate():
    global grid
    a=check()
    wx, wy = a[0]
    bx, by = a[1]
    Wc, Bc = a[2], a[3]
    if Wc:
        for x in [0,1,-1]:
            for y in [0,1,-1]:
                if wx + x in range(8) and wy + y in range(8) and grid[wx+x][wy+y]=='.':
                    a=check(wx+x,wy+y,-1,-1,mode='W')
                    if a==False:
                        return False,1
        print("CheckMate for White")
        return (True,'W')
    elif Bc:
        print("Boxes Checked:")
        for x in [0,1,-1]:
            for y in [0,1,-1]:
                if bx+x in range(8) and by+y in range(8) and grid[bx+x][by+y]=='.':
                    print(bx+x,by+y)
```

Fig 5.4

```
                    a=check(bx+x,by+y,-1,-1,mode='B')
                    if a==False:
                        return False,1
        print("CheckMate for Black")
        return (True,"B")
    else:
        return False,

def straight(x,y,ex,ey):
    if (x==ex or y==ey) and obstructed(x,y,ex,ey):
        return True
    return False

def diagonal(x,y,ex,ey):
    if abs(x-ex)==abs(y-ey) and obstructed(x,y,ex,ey):
        return True
    return False
```

Fig 5.5

```
def lmove(x,y,ex,ey):
    if (abs(x-ex),abs(y-ey)) in [(2,1),(1,2)]:
        return True
    return False

def movable(coin,x,y,ex,ey):
    color = coin[0]
    coin=coin[1]
    if(coin=='P'):

        if (straight(x,y,ex,ey) or diagonal(x,y,ex,ey)) and (abs(y-ey)<=1 and abs(x-ex)==1):
            print("Got here!", color)
            if color=='B' and ex>x :
                return True
            elif color=='W' and ex<x:
                return True
            else:
                return False
        else:
            return False

    elif coin=='Q':
        if straight(x,y,ex,ey) or diagonal(x,y,ex,ey):
            return True
        return False

    elif coin=='R':
        if straight(x,y,ex,ey):
            return True
        return False
```

Parameter x of src.code.movable
x: Any

Fig 5.6

```
    elif coin=='B':
        if diagonal(x,y,ex,ey):
            return True
        else:
            return False

    elif coin=='H':
        return lmove(x,y,ex,ey)

    elif coin=='K':
        if (straight(x,y,ex,ey) or diagonal(x,y,ex,ey) ) and (abs(x-ex)<=1 and abs(y-ey)<=1):
            return True
    else:
        return False
```

Fig 5.7

```
def validate_move(gr,t,x,y,ex,ey):
    global turn,grid
    turn=t
    grid=gr
    coin=grid[x][y]
    end_coin=grid[ex][ey]
    # 'WK'
    if coin=='.' or (turn==0 and (coin[0]=='B' or end_coin[0]=='W')) or (turn==1 and (coin[0]=='W' or end_coin[0]=='B')):
        return False
    else:
        res=movable(coin,x,y,ex,ey)
        if res is True:
            print("True part!")
            return True
        else:
            print("Invalid move...Given end not reachable!")
            return False

def move():
    turn=0
    if(turn==1):
        print("White's move:")
        x,y=map(int,input("Enter start point").split(' '))
        ex,ey=map(int,input("Enter start point").split(' '))
        res=validate_move(x,y,ex,ey)
        turn=0
    else:
        print("Black's move")
        x, y =map(int,input("Enter start point").split(' '))
```

Fig 5.8

```
        x,y=map(int,input("Enter start point").split(' '))
        ex,ey=map(int,input("Enter start point").split(' '))
        res=validate_move(x,y,ex,ey)
        turn=0
    else:
        print("Black's move")
        x, y =map(int,input("Enter start point").split(' '))
        ex, ey =map(int,input("Enter start point").split(' '))
        res = validate_move(x, y, ex, ey)
        turn=1

def set_grid(gr):
    global grid
    gride=gr

#init_grid()
#while(True):
    #move()
    #print_grid()
```

Fig 5.9

5.2 GUI code for the chess board

```
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtCore import QSize
from PyQt5.QtGui import QIcon, QPixmap
from PyQt5.QtWidgets import QPushButton

from src.Mod_button import ModButton
from src.code import validate_move,check,check_mate,movable,set_grid

class Ui_Dialog(QtWidgets.QMainWindow):
    def setupUi(self, Dialog):
        Dialog.setObjectName("Dialog")
        Dialog.resize(1263, 873)
        self.points=[]
        self.BKStyle,self.WKStyle=None,None
        self.turn=0
        self.modified=[]
        self.gridLayoutWidget = QtWidgets.QWidget(Dialog)
        self.gridLayoutWidget.setGeometry(QtCore.QRect(100, 60, 800, 800))
        self.gridLayoutWidget.setObjectName("gridLayoutWidget")
        self.gridLayout = QtWidgets.QGridLayout(self.gridLayoutWidget)
        self.gridLayout.setContentsMargins(0, 0, 0, 0)
        self.gridLayout.setObjectName("gridLayout")
        self.gridLayout.setHorizontalSpacing(0)
        self.gridLayout.setSpacing(0)
        self.gridLayout.setVerticalSpacing(0)

        self.retranslateUi(Dialog)
        QtCore.QMetaObject.connectSlotsByName(Dialog)
        self.init_grid()
```

Fig 5.10

```
        self.add_buttons()

    def init_grid(self):
        grid=[(['.']*8)[:] for x in range(8)]
        arr = ['R', 'H', 'B', 'K', 'Q', 'B', 'H', 'R']
        for x in range(8):
            grid[0][x] = 'B' + arr[x]

        for x in range(8):
            grid[-1][x] = 'W' + arr[x]

        for x in range(8):
            grid[1][x] = 'BP'

        for x in range(8):
            grid[-2][x] = 'WP'

        self.grid=grid
```

Fig 5.11

```
def render_board(self):
    for x in range(8):
        for y in range(8):
            if self.grid[x][y]!='.':
                self.buttons[(x,y)].setIcon(QIcon(f"C:\\Users\\Karthik Surya J\\PycharmProjects\\chess_clashofkings\\src\\
                self.buttons[(x,y)].setIconSize(QSize(70,70))
            else:
                self.buttons[(x, y)].setIcon(QIcon())


def add_buttons(self):
    self.buttons={}
    btn=self.buttons
    cnt=0
    for x in range(8):
        cnt+=1
        for y in range(8):
            btn[(x,y)]=ModButton(self)
            btn[(x,y)].add_x_y(int(x),int(y))_# tell to change this int(thin)
            btn[(x,y)].clicked.connect(self.called(x,y))
            self.gridLayout.addWidget(btn[(x,y)],x,y)
            btn[(x, y)].setFixedSize(100, 100)
            style1,style2="background-color:rgba(255,255,255,255);border:none;","background-color:rgba(0,0,110,255);borde
            if (y+cnt)%2==0:
                btn[(x,y)].setStyleSheet(style1)
            else:
                btn[(x,y)].setStyleSheet(style2)
    print(self.buttons)
```

Fig 5.12

```
    self.render_board()

def called(self,x,y):
    def clicked():
        self.updated(x,y)

    return clicked

def updated(self,x,y):
    if len(self.points)==0:
        self.points.append((x,y))
        set_grid(self.grid)
        self.show_possible_moves(x,y)
    else:
        self.undo_selected()
        self.points.append((x,y))
        x1,y1=self.points[0][0],self.points[0][1]
        x2, y2 = self.points[1][0], self.points[1][1]
        res=self.check_valid()
        # res=True
        self.points=[]
        print("Res valuse is",res)
        if res is True:
            self.turn=(self.turn+1)%2
            self.grid[x2][y2]=self.grid[x1][y1]
            self.grid[x1][y1]='.'
            self.checker()
            self.render_board()
        else:
            print("Wrong move!!")
```

Fig 5.13

```
    def check_valid(self):
        # points =[(1,2),(2,3)]
        x1,y1=self.points[0][0],self.points[0][1]
        x2, y2 = self.points[1][0], self.points[1][1]
        print(x1,y1,x2,y2)
        res=validate_move(self.grid,self.turn,x1,y1,x2,y2)
        print("res is ",res)
        return res

    def checker(self):
        a=check()
        self.king_coord=a[0],a[1]
        wx,wy=a[0]
        bx,by=a[1]
        Wc,Bc=a[2],a[3]
        print("Check Value is ",Wc,Bc)
        if Wc:
            self.WKStyle=self.buttons[(wx,wy)].styleSheet()
            self.buttons[(wx,wy)].setStyleSheet('background-color:red;border:none;')
        else:
            if self.WKStyle is not None:
                self.buttons[(wx,wy)].setStyleSheet(self.WKStyle)
        if Bc:
            self.BKStyle = self.buttons[(wx, wy)].styleSheet()
            self.buttons[(bx, by)].setStyleSheet('background-color:red;border:none;')
        else:
            if self.BKStyle is not None:
                self.buttons[(wx,wy)].setStyleSheet(self.BKStyle)
```

Fig 5.14

```
    def show_possible_moves(self, x, y):
        gr = self.grid
        for ex in range(8):
            for ey in range(8):
                if self.grid[ex][ey] == '.':
                    if movable(gr[x][y], x, y, ex, ey):
                        self.modified.append((self.buttons[(ex, ey)], self.buttons[(ex, ey)].styleSheet()))
                        self.buttons[(ex, ey)].setStyleSheet("background-color:rgb(186, 202, 69);border:none;")
                    else:
                        self.buttons[(ex, ey)].setIcon(QIcon())

    def undo_selected(self):
        for x in self.modified:
            x[0].setStyleSheet(x[1])

    def retranslateUi(self, Dialog):
        _translate = QtCore.QCoreApplication.translate
        Dialog.setWindowTitle(_translate("Dialog", "Dialog"))

app = QtWidgets.QApplication([])
w = Ui_Dialog()
w.setupUi(w)
w.show()
import sys
sys.exit(app.exec_())
```

Fig 5.15

CHAPTER 6

RESULTS

6.1 Chess board

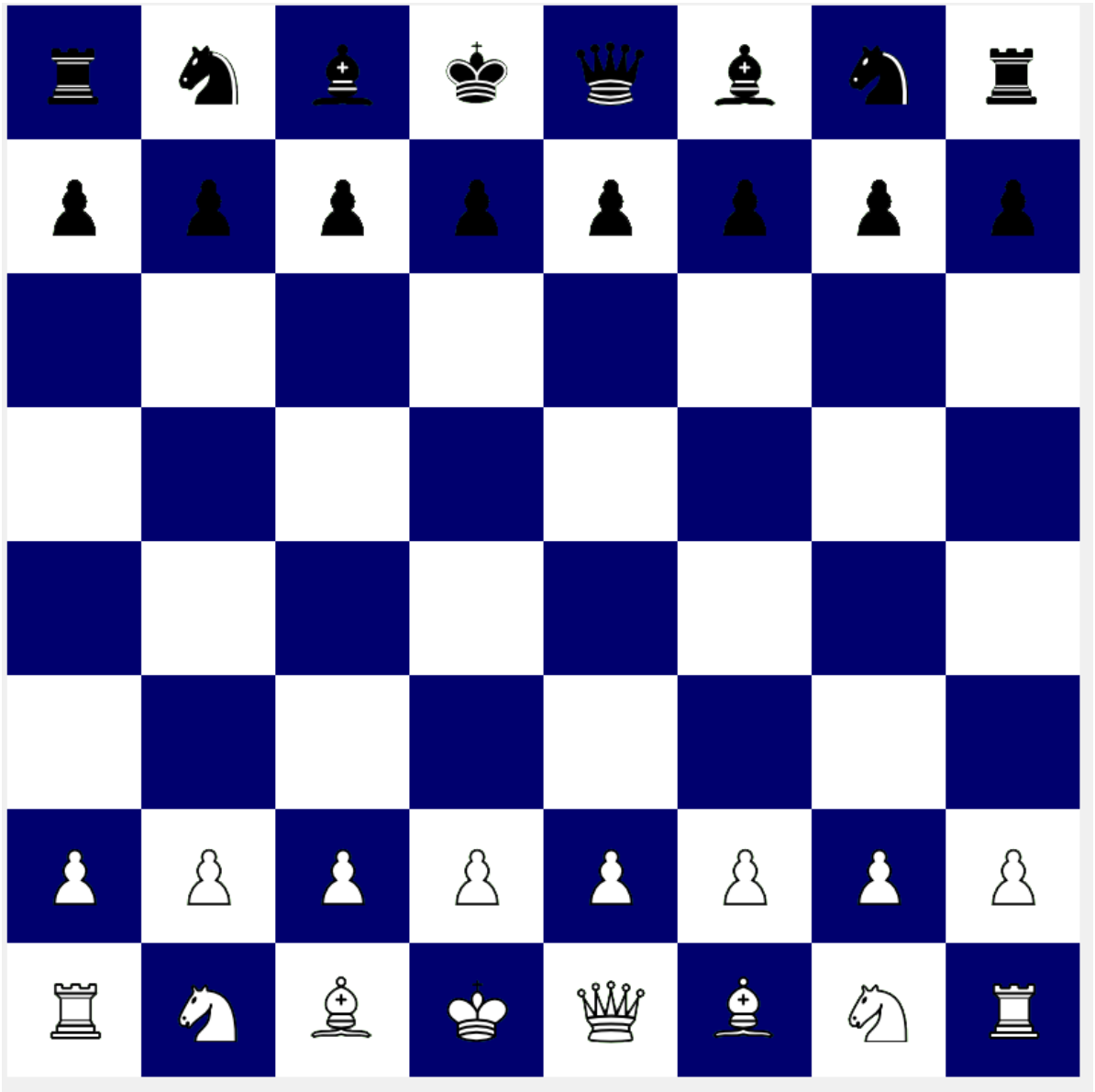

Fig 6.1

6.2 Displays all the possible moves

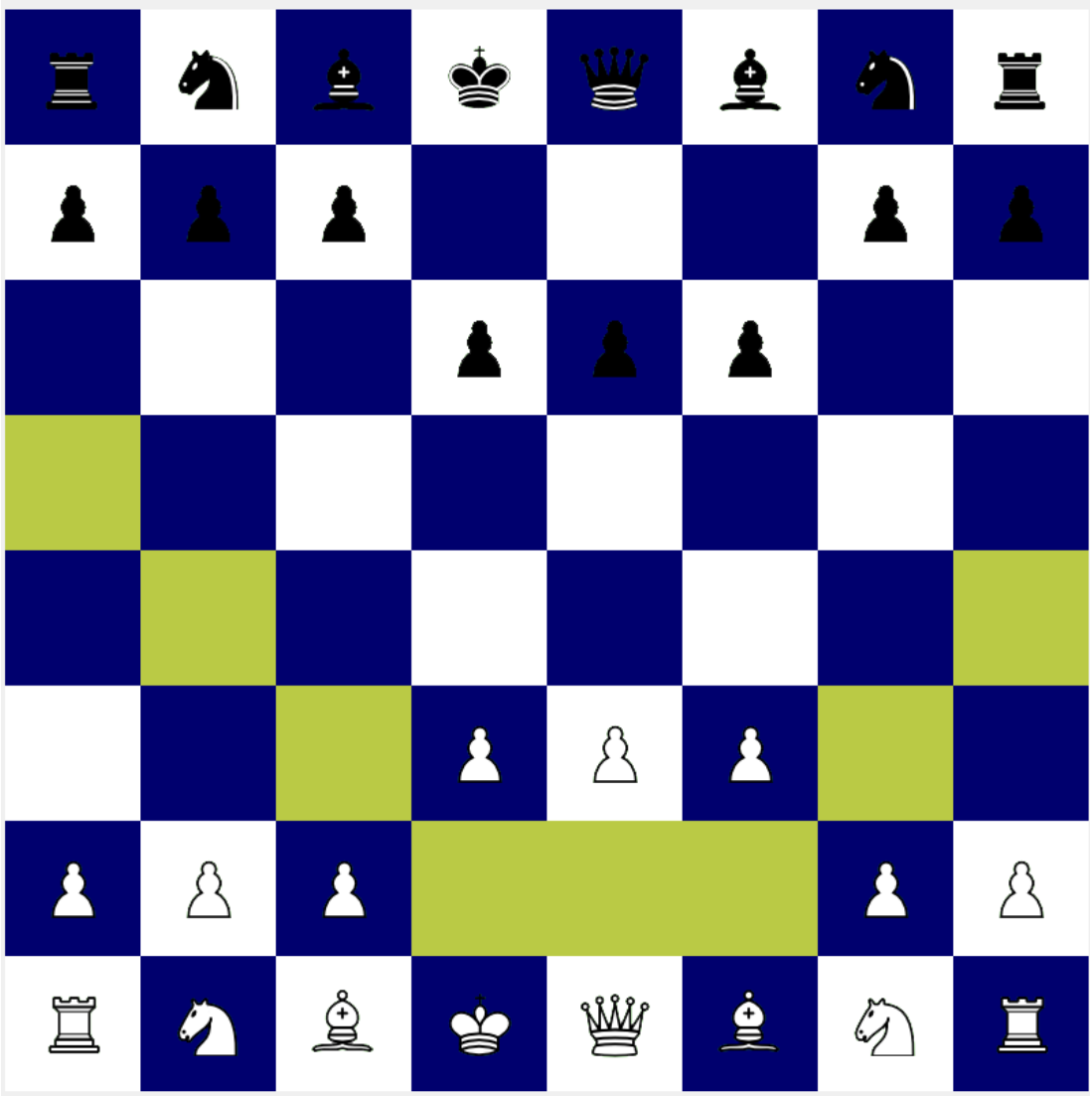

Fig 6.2

6.3 CHECK CONDITION

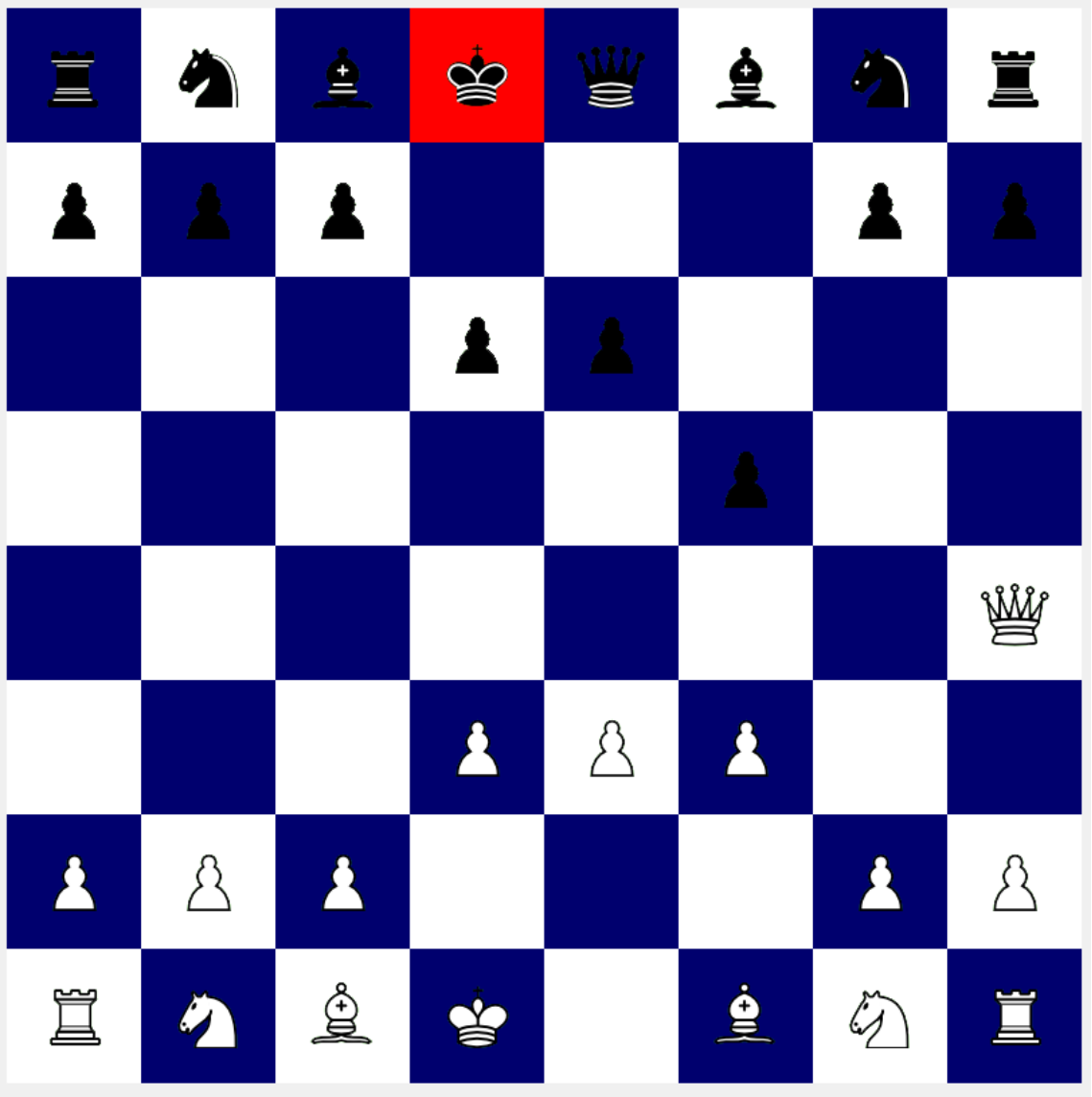

Fig 6.3

CHAPTER 7

CONCLUSION

This mini project has successfully accomplished the goals it had set out in the objectives and design sections of this report. I have also learnt many new concepts of python and have used and implemented successfully, and pyqt5 platform gui toolkit provides many options and easy to understand.

Chess strategy focuses on setting and achieving long-term goals during the game (such as where to put other pieces), while tactics focus on immediate maneuvers.

These two parts of chess thinking cannot be completely separated. This is because strategic objectives are primarily achieved through tactics and tactical abilities are based on previous strategies in the game.

Chess strategy is about evaluating your chess positions and setting long-term plans for your goals and future. During the evaluation, the player must consider many factors such as the value of the pieces on the board, central control and centralization, pawn structure, king safety, and control of key squares or groups of squares (eg diagonals, open files), etc.

In chess, tactics usually focus on short-term actions that can be calculated in advance by a human or computer player. The possible depth of computation depends on the player's abilities or processor speed. Deep calculations are impossible in a calm position with many possibilities on either side, but in a "tactical" position with a limited number of coercion options, far fewer than best moves can be lost quickly, so strong players can expect very long sequences of actions there are moves.

CHAPTER 8

REFERENCES

[Boehm81] Barry Boehm. Copyright © 1981. 0138221227. Prentice-Hall PTR. *Software Engineering Economics*.

[Comer95] Douglas Comer. Copyright © 1995. 3rd edition. 0132169878. Prentice-Hall. *Internetworking with TCP/IP*. Vol. I. Principles, Protocols, and Architecture.

[Cormen90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. Copyright © 1990. 0262031418. MIT Press. *Introduction To Algorithms*.

[Dijkstra76] Edsger Dijkstra. Copyright © 1976. 0613924118. Prentice-Hall. *A Discipline of Programming*.

[Gries81] David Gries. Copyright © 1981. 0387964800. Springer-Verlag. *The Science of Programming*.

[Holt78] R. C. Holt, G. S. Graham, E. D. Lazowska, and M. A. Scott. Copyright © 1978. 0201029375. Addison-Wesley. *Structured Concurrent Programming with Operating Systems Applications*.

[Knuth73] Donald Knuth. Copyright © 1973. 0201896834. Addison-Wesley. *The Art of Computer Programming*. Volume 1. Fundamental Algorithms.

[Meyer88] Bertrand Meyer. Copyright © 1988. 0136290493. Prentice Hall. *Object-Oriented Software Construction*.

[Parnas72] D. Parnas. Copyright © 1972. Communications of the ACM. *On the Criteria to Be Used in Decomposing Systems into Modules*. 5. 12. December 1972. 1053-1058.

[SEIstr04] Software Engineering Institute. Copyright © 2004. https://www.sei.cmu.edu/str/descriptions/index.html. *Software Technology Roadmap*.