Name= Himanshu Yadav

Roll no =A05

K18je

11807655

Abstract:

Chess which do play between two players on a board is intellectual and

mental game, it has its own rules of play which help to enhance and im-

prove the mental and intellectual activities of the player, and this game

has a huge amount of players around the all world they have strongly

interested to have play it. This document deals with the fully computer-

ized Chess Game, \_rst, the game computerizes for two player to do play

chess according all the valid rules of the chess on computer. Secondly,for

making the game more interesting that will make users to direct do play

against computer, computer intellectual force is added.

Introduction:

Chess is a game for two players, dubbed White and Black. The goal is to

capture your opponent's king. In the game, this is known as a checkmate.Chess

is played on a board with 64 squares. Each player begins with 16 pieces, lined

up in two rows. The \_rst row is occupied by pieces called pawns. The next

row contains: a king, a queen, two rooks, two bishops, and two knights.

Chess is de\_ned as a game of \perfect information", because both players

are aware of the entire state of the game world at all times: just by looking

at the board, you can see which pieces are alive and where they are located.

Checkers, Go, Go-Moku,Backgammon and Othello are other members of the

category, but stud poker is not (you don't know what cards your opponent is

holding in his hands).

Here that to able to change chess game from physical form to \_gurative form

fully realistic, Several things are needed to make chess game computerized and

intelligent.

Literature review:

For making computerized chess game, which will let two players to play chess

game in computer realistic. We need two things which we must introduce to

computer.

\_ Some way to represent a chess board in memory, so that it knows what

the state of the game is.

\_ Rules to determine how to generate legal moves, so that it can play

without cheating (and verify that its human opponent is not trying to

pull a fast one on it !).

First we should introduce chess board and its elements to computer, next we

need to produce movement for chess pieces according to rule if chess game.

Proposed methodology:

This code is in python where we have implemented the method of A\* method and also a method named as heuristic which returns the manhattan distance that is the difference of a grid from present location to its desired location in goal state and a method named as moves which returns the set of all possible moves in a scenario

To a computer, it is far from obvious which of many legal moves are "good"

and which are "bad". The best way to discriminate between the two is to look

at their consequences (i.e., search series of moves, say 4 for each side and look

at the results.) And to make sure that we make as few mistakes as possible,

we will assume that the opponent is just as good as we are. This is the basic

principle underlying the minimax search algorithm, which is at the root of all

chess programs.

Unfortunately, minimax' complexity is O(bn), where b ("branching factor")

is the number of legal moves available on average at any given time and n (the

depth) is the number of "plies" you look ahead, where one ply is one move

by one side. This number grows impossibly fast, so a considerable amount of

work has been done to develop algorithms that minimize the e\_ort expended

on search for a given depth. Iterative-deepening Alphabeta, NegaScout and

MTD(f) are among the most successful of these algorithms, we will discuses

with more detail later in this document.

Another major source of headaches for chess programmers is the "horizon

e\_ect", \_rst described by Hans Berliner. Suppose that your program searches

to a depth of 8-ply, and that it discovers to its horror that the opponent will

capture its queen at ply 6. Left to its own devices, the program will then

proceed to throw its bishops to the wolves so that it will delay the queen

capture to ply 10, which it can't see because its search ends at ply 8. From the

program's point of view, the queen is "saved", because the capture is no longer

visible... But it has lost a bishop, and the queen capture reappears during the

next move's search. It turns out that \_nding a position where a program can

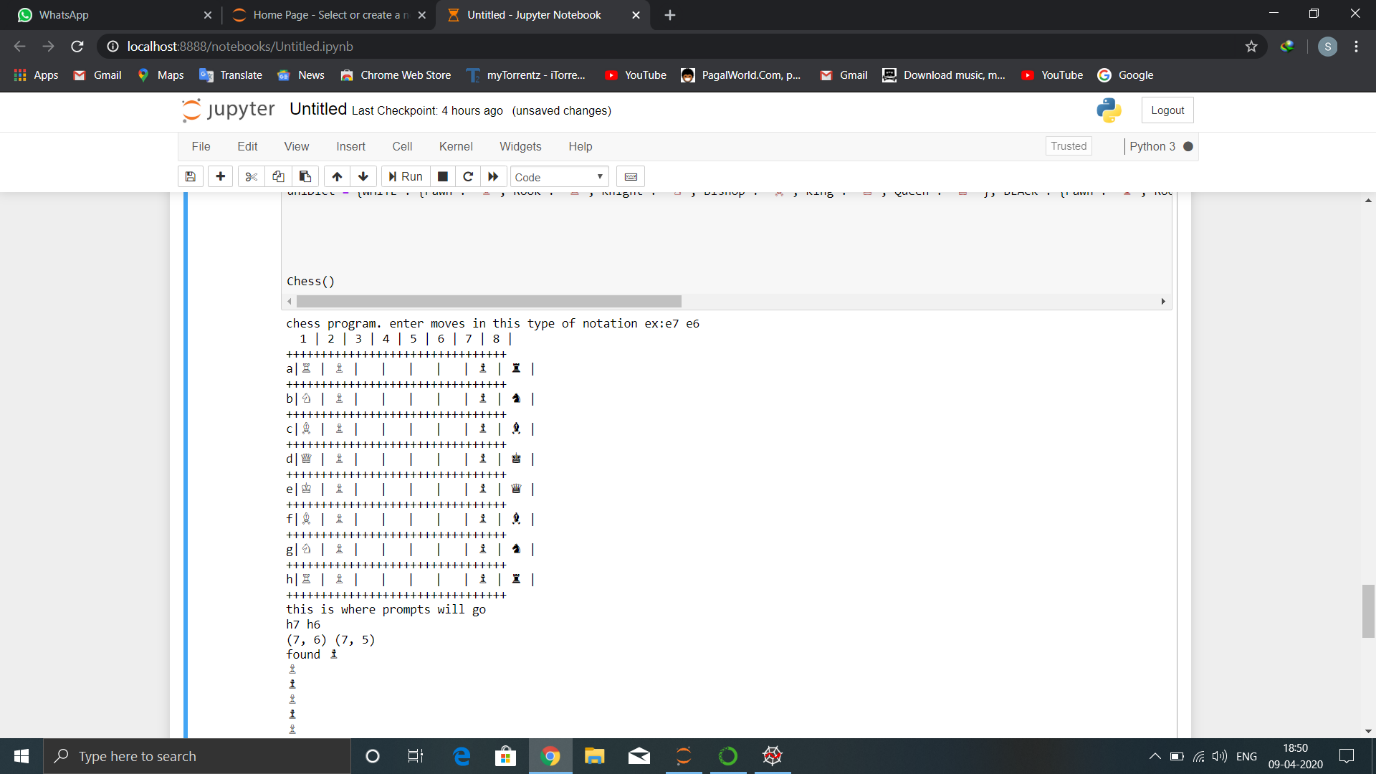
reason correctly about the relative strength of the forces in presence is not a

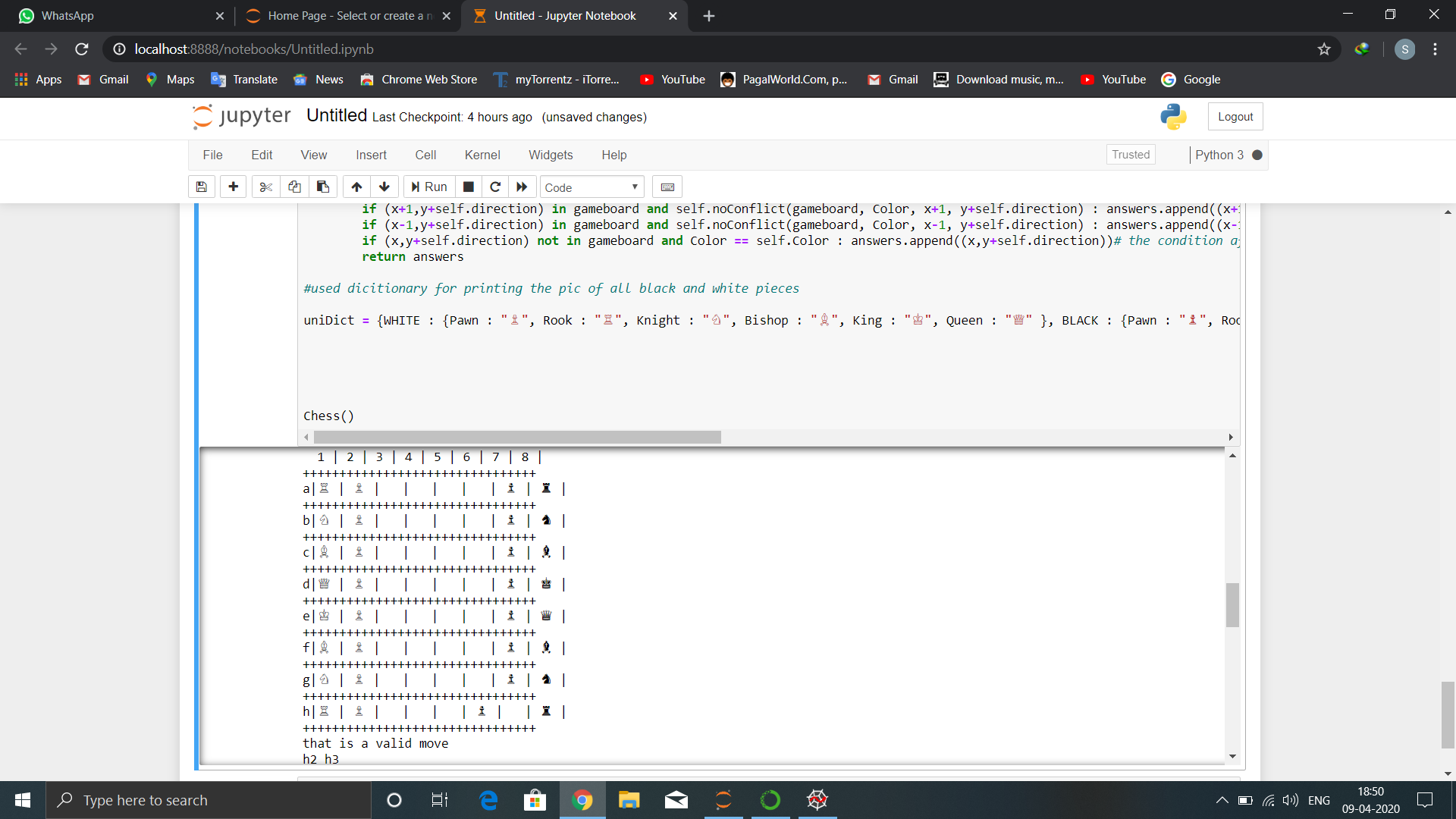
trivial task at all, and that searching every line of play to the same depth is

tantamount to suicide. Numerous techniques have been developed to defeat

the horizon e\_ect.

Result and discussion:





Conclusion:

This project was really helpful in building up the ability to think critically about the algorithm which we used in the project and also that time can be saved by implementing algorithms for solving problems rather than solving it manually, however building this code required certain perquisites like possible moves in the given scenario, rules for solving the Chess etc. The scope of artificial intelligence is very large in today’s era where machines are taking over difficult jobs and thereby increasing the productivity.

References:

Python crash course book by Eric Matthes

Python essential reference book by M.Beazley