**Minor Project Report**

**On**

**A Multiplayer Chess**

**Submitted by:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Roll No** | **Branch** |
| Sarthak Jain | R171218088 | CSE-DevOps |
| Muskaan Madan | R171218063 | CSE-DevOps |
| Siddharth Chopra | R178218045 | CSE-MAD |
| ManviKulshrestha | R178218022 | CSE-MAD |

**Under the guidance of**

Dr. Monit Kapoor

Associate Professor

HOD - Department of Cybernetics



**School of Computer Science and Engineering**

**University of Petroleum & Energy Studies**

**Dehradun - 248001**

**2020**

**INDEX**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **TITLE** | **PAGE NO.** |
| 1 | Project Title and Abstract | 3 |
| 2 | Introduction | 4 |
| 3 | Problem Statement | 5 |
| 4 | Literature Review | 6-7 |
| 5 | Objective | 8 |
| 6 | Methodology | 9 |
| 7 | System Requirement | 10 |
| 8 | Pert Chart | 11 |
| 9 | Work Done So Far | 12-19 |
| 10 | References | 20 |



**School of Computer Science and Engineering**

**University of Petroleum & Energy Studies, Dehradun**

**Project Proposal Approval Form (2020)**

**PROJECT TITLE:**

**A Multiplayer Chess**

**ABSTRACT:**

In this project we aim to implement the standard chess game using C, a basic and a broadly usedlanguage, supporting structured programming. The chessboard will be 2D and will have a grid size of 8 \* 8. The chessboard and the pieces are going to be the essential elements of the project, on which further, we will be implementing thelogic of the game. Without these two parts, all the functionalities cannot be achieved as the board must restrict the movement of pieces at any time, and the game cannot run without the pieces.[1]

The game provides human to human capability for both playing and learning, in which they need to be present in the same geographical location physically. It is a offline game which asks the user to choose their set of color for further playing and thus helps in deciding who wants the make the first move.

need not to be present in the same

geographical location physically.

**Keywords**: chessboard and pieces, human to human capability.

1. **INTRODUCTION**

In the last five years the strength of computer chess programs has grown immensely. They have a place with easygoing games, and it requires two players to continue on a chessboard of size 8 \* 8. The guidelines of this game are straightforward yet have boundless ways to move pieces, thus each step is urgent to the player.

Initially we are working with 2 player game. Later we will step up with multiplayers with similar algorithm. This program for chess will be the most efficient one till now in C language. The program supports a reasonable number of functions and the game will be in 2D. This game will be played in same system by the two different players.

# **PROBLEM STATEMENT:**

How might we create new computer chess model so that chess engines merge the prerequisites on knowledge implementation with a maximum of efficiency?

Our challenge is to develop multiplayer chess, for which we require three things:

* First is the chess knowledge with the unlimited implementation,
* Second is the higher performance or the higher computing speed,
* The last is to minimize the unnecessary overhead.

1. **LITERATURE REVIEW:**

A relationship between chess skill and intelligence has long been discussed in the literature and popular culture. This involves having the whole knowledge of chess including its special as well as the basic rules. Some of the special rules include the En passant rule, which is a special pawn capturing move in chess. Pawns can usually capture only pieces that are directly and diagonally in front of it on an adjacent file. It moves to the captured piece's square and replaces it. With en passant, though, things are a little different.

This type of capture is the only one in chess where the capturing piece doesn't land on the same square as its victim. To perform this capture, you must take your opponent's pawn as if it had moved just one square. You move your pawn diagonally to an adjacent square, one rank farther from where it had been, on the same file where the enemy's pawn is, and remove the opponent's pawn from the board. [2] Occasionally chess games do not end with a winner, but with a draw. One of the reasons why a chess game may end in a draw is, Fifty moves. This means that, when fifty consecutive moves have been played where neither player has moved a pawn or captured a piece, the game gets draw.[3]

**Optimization** is about to choose the best element from some set of available alternatives. Most importantly there is the algorithmic optimization on design level such as using **alpha-beta** rather than plain minimax, followed by source code optimizations using searches such as using **Quiescence Search**- whose purpose is to only evaluate "quiet" positions, or positions where there are no winning tactical moves to be made. This search is needed to avoid the horizon effect. Simply stopping your search when you reach the desired depth and then evaluate, is very dangerous. [4] The other important parts of the search are: Null Move Pruning and Check Extension.

So, **Null Move Pruning**, is a method based on the Null Move Observation to reduce the search space by trying a "null" or "passing" move, then seeing if the score of the subtree search is still high enough to cause a beta cutoff. Nodes are saved by reducing the depth of the subtree under the null move. [5] Where as **Check Extensions** have two distinct forms: one of them extends when giving check, the other - when evading it. In each case, typical depth to extend is one ply. The reason behind check extension is that we are in a forcing sequence, so that it is desirable to know its outcome with more certainty, and the number of replies to check is limited, so we do not have to be afraid of a search explosion. Also, not extending checks may easily lead to the horizon effect, delaying the threat so far that the program cannot see it. [6]

After source code optimization we finally have compiler optimization. Some other concepts or factors which help in increasing the efficiency of the chess code are:

**Forsyth-Edwards Notation** (FEN), is used to describe exactly one static position on the chessboard. Valid FEN strings have strict formatting requirements so that they can be easily parsed by a program. It is nice if you want to represent the whole game in one single string without any need to know the entire game.

**Other than that** the modern engines handle positions using **bitboards**. **Bitboards**are how engines internally represent positions, and are likely to be much less familiar than FEN. Locations marked with '1' contain the piece; locations marked with '0' do not contain the piece.

One reason bitboards are useful is that they can be used with bitwise logical operators. These are functions like AND, OR, NOT, etc., which are all extremely fast to execute on a CPU- significantly faster than simple addition. Furthermore, 64-bit processors can complete 64-bit logical operations in one CPU cycle. This has obvious advantages for a 64-square chessboard, for which each bitboard is exactly 64-bits long. [7]

The next we have **Principal variation** (PV) which is a sequence of moves that programs consider best and therefore expect to be played. All the nodes included by the PV are PV-nodes. [8]

The next is **Transposition Table**, it is a database that stores results of previously performed searches. It is a way to greatly reduce the search space of a chess tree with little negative impact. [9] It will help while searching and ordering moves, giving a brutal difference in speed.

**History Heuristic**, a dynamic move ordering method based on the number of cutoffs caused by a given move irrespectively from the position in which the move has been made. [10] And **Performance Testing** which is a software testing process used for testing the speed, response time, stability, reliability, scalability and resource usage of a software application under particular workload. The main purpose of performance testing is to identify and eliminate the performance bottlenecks (problem causing delays to process) in the software application. It is a subset of performance engineering and also known as “Perft Testing”. Without this, software is likely to suffer from issues such as: running slow while several users use it simultaneously, inconsistencies across different operating systems and poor usability. [11]

Perft can receive another speed boost by [hashing](https://www.chessprogramming.org/Hash_Table) node counts, with a small chance for inaccurate results. Sometimes this is used as a sanity check to make sure the keys are working correctly.

.

1. **OBJECTIVE:**

The objective of our project is to create a multiplayer chess engine to adhere to efficiency.

**Sub-objective incidental to main objective are-**

1. Identification of the rules to be implemented.
2. Designing and creating the 8 \* 8 chess-board with pieces.
3. Designing and implementing the algorithms for fundamental and special rules.
4. **METHODOLOGY:**

First we will create the basic structure of 8x8 chess board.

Basically array of 120 will be made instead of 64 so that we check whether the pieces are going out or not.

The list of all the pieces will be made (8 pawns, 2 rooks, 2 knights, 2 bishops, a queen and the most important one – the king) for both the colors.

For each pieces there will be a unique function with its algorithm to move.

The code contains a separate algorithm to check which player have to take the move next.

An algorithm will be made for one of the rule – CASTLING. Two separate algorithms will be created for two different type castling.

* **King-side castling** – where the White king goes two spaces to his right, and on the other side of the board the Black king can go two spaces to his left.
* **Queen-side castling** – similar in that the king moves two spaces but this time the White king goes left and the Black king goes right.

A fifty move rule will also be checked.

Another algorithm will be created to check whether castling can be applied or not. This rule is applied only when king has not taken a single move.

Player can undo his last move, so a function will be made for the same.

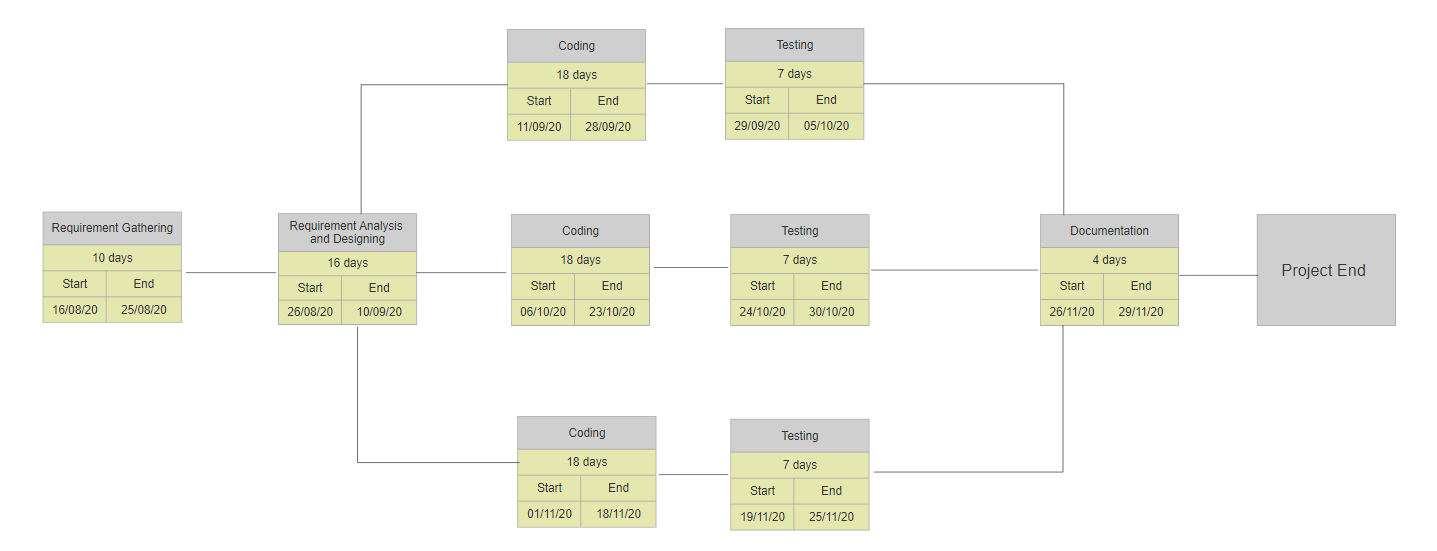
If the checkmate formula occurs, End Game.

1. **SYSTEM REQUIREMENTS:**

Our program empowers two clients to play chess, which implies there is no AI engaged with our work. The game needs to make various pieces, imagine the chessboard, acknowledge different clients input controls, and know when to end the game cycle.

* Hardware Requirements
* Computer system
* Minimum 8GB RAM
* Software Requirements
* Any OS i.e. Windows, Linux and macOS
* Any text editor such as Notepad++ and Console
* Internet Access

1. **PERT CHART**

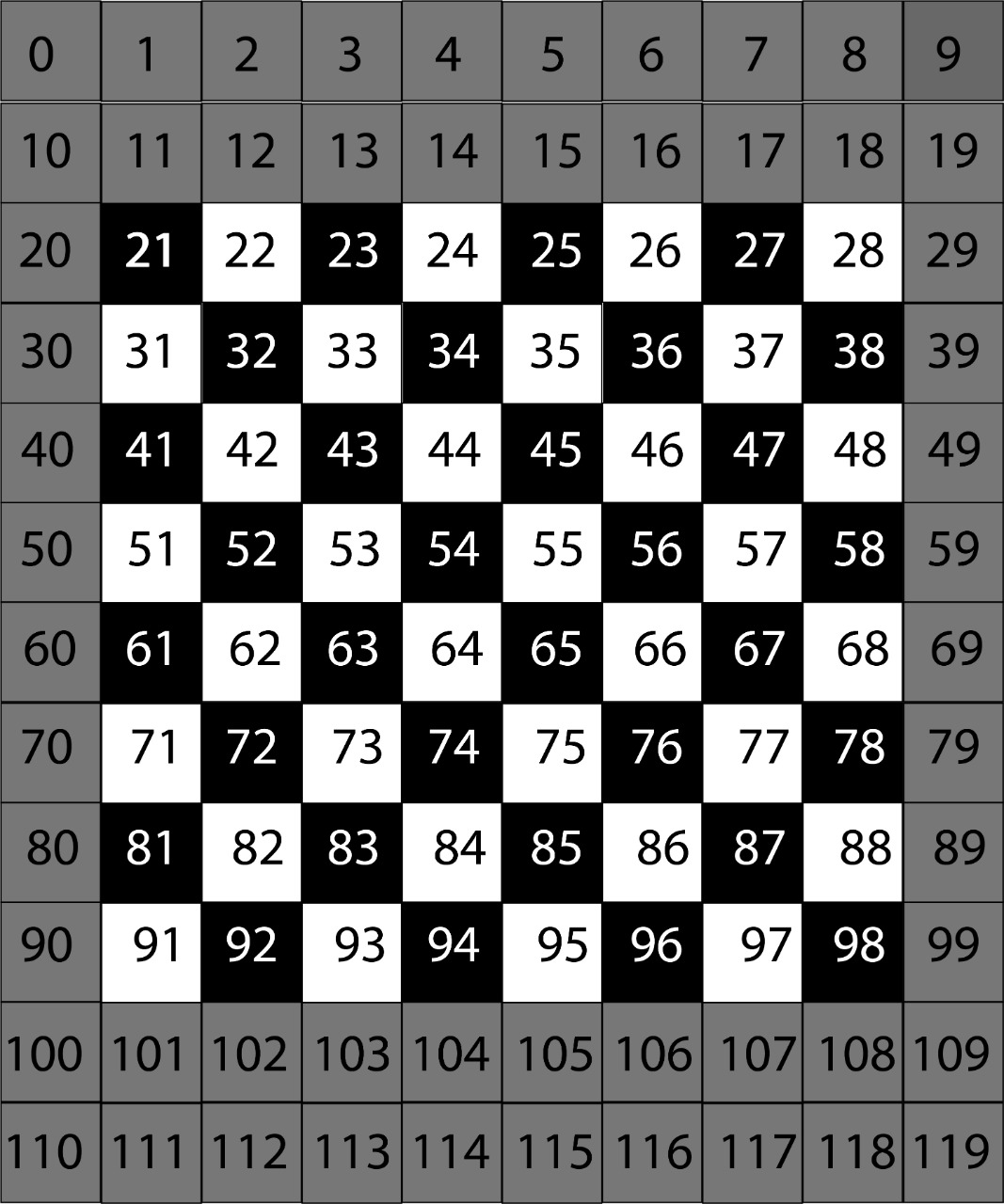


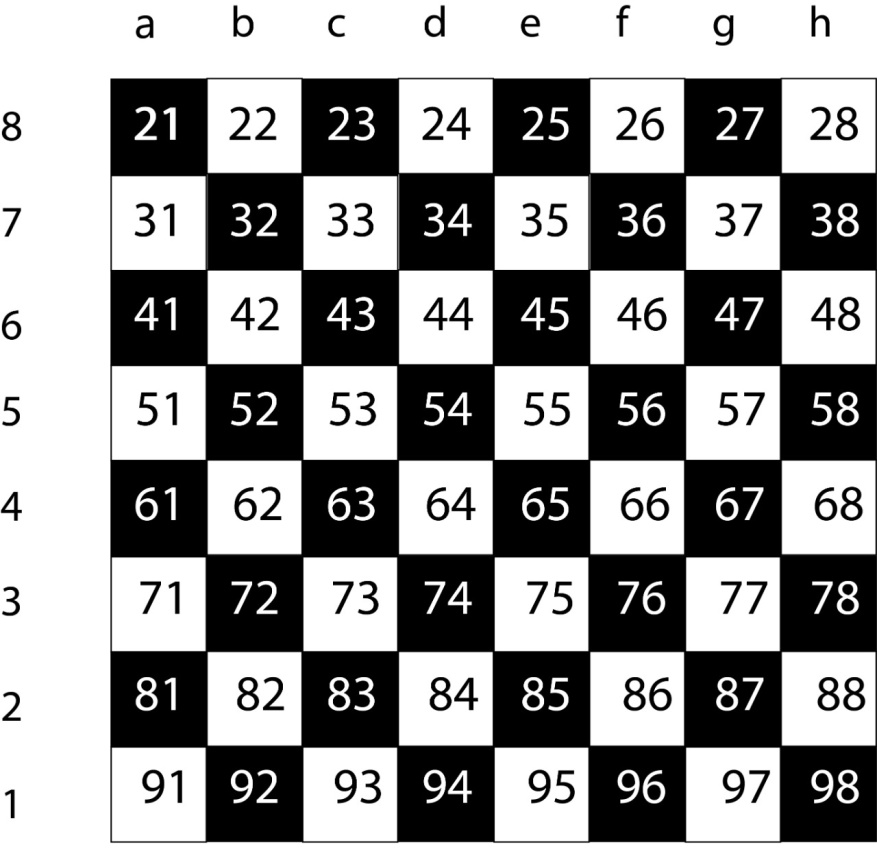
1. **WORK DONE SO FAR**

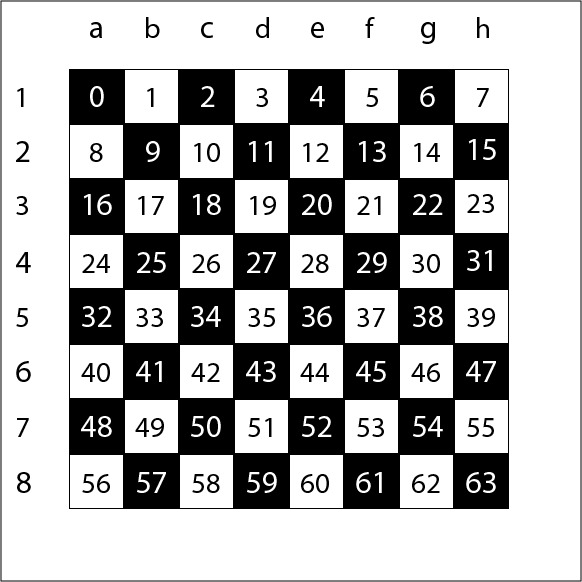
We are prepared with the main board structure so far. There will be a board of 64 bits that will work on U64 type. The board is printed with the pieces and the following rules are also in the implementation:

1. UNDO – Move function.
2. Generation of position key
3. Permission of castling has been implemented
4. And fifty move rule has also been implemented.

**Diagrams:**

****

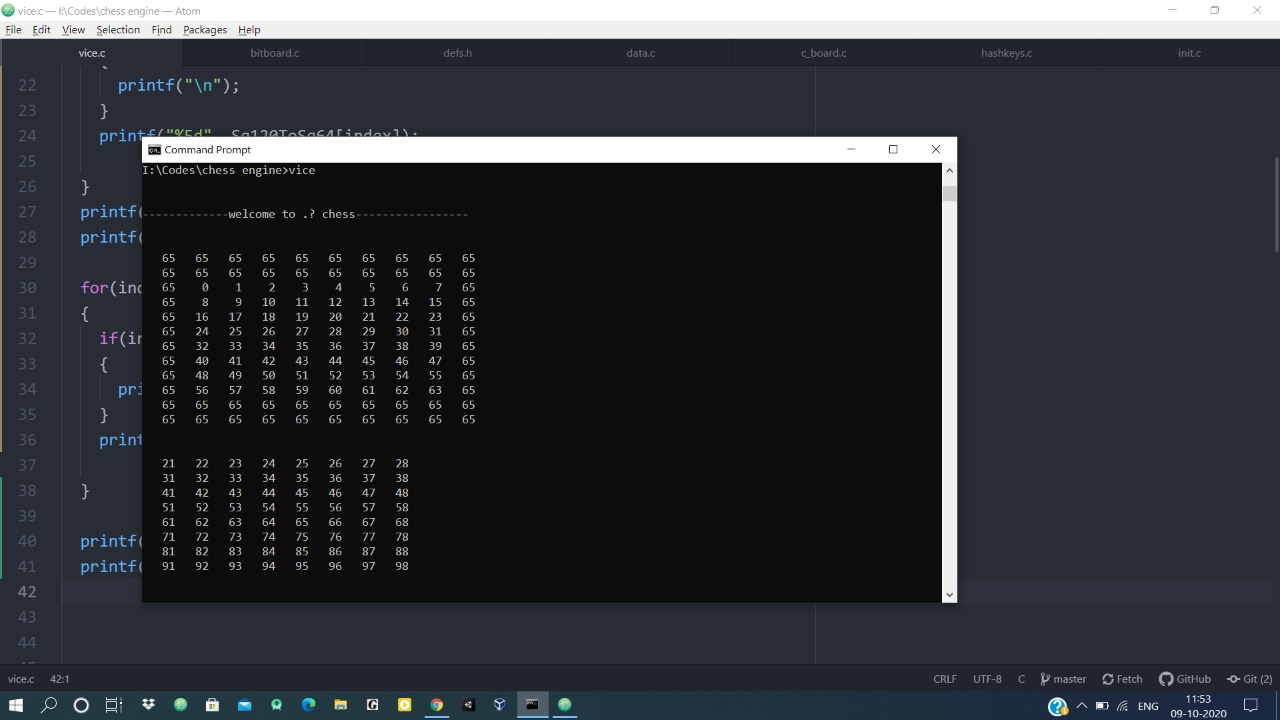
****

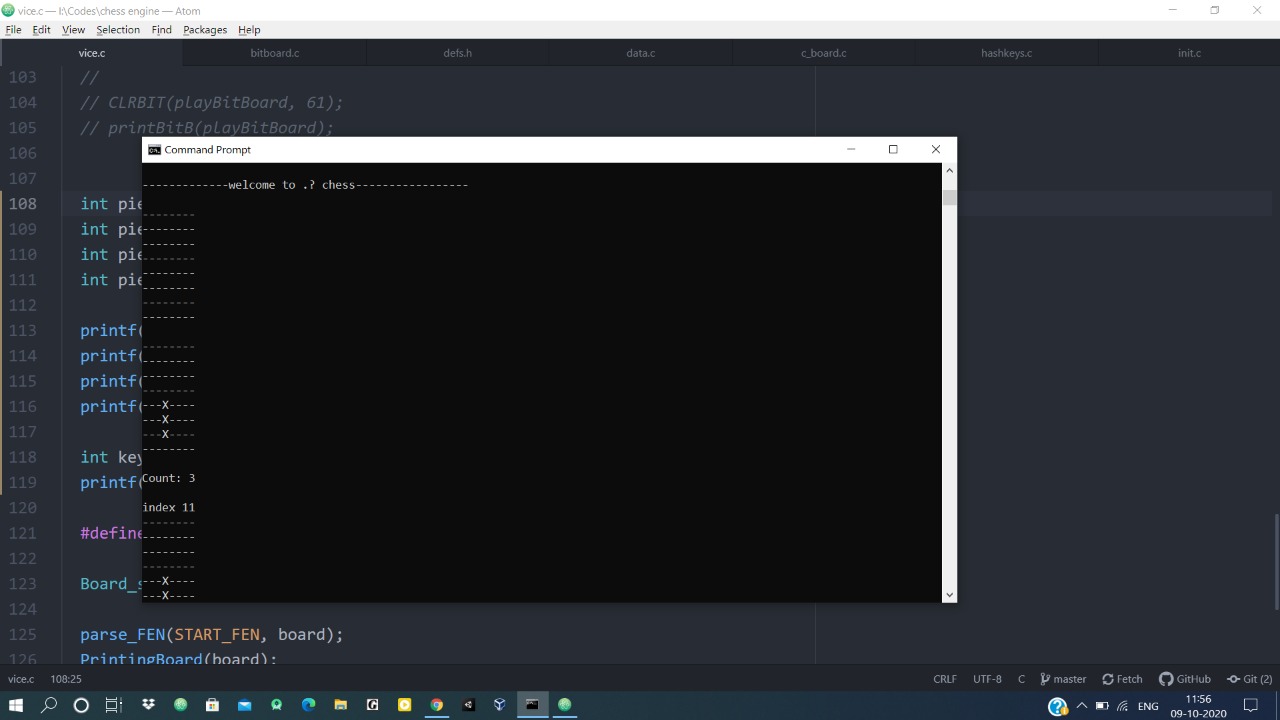
****

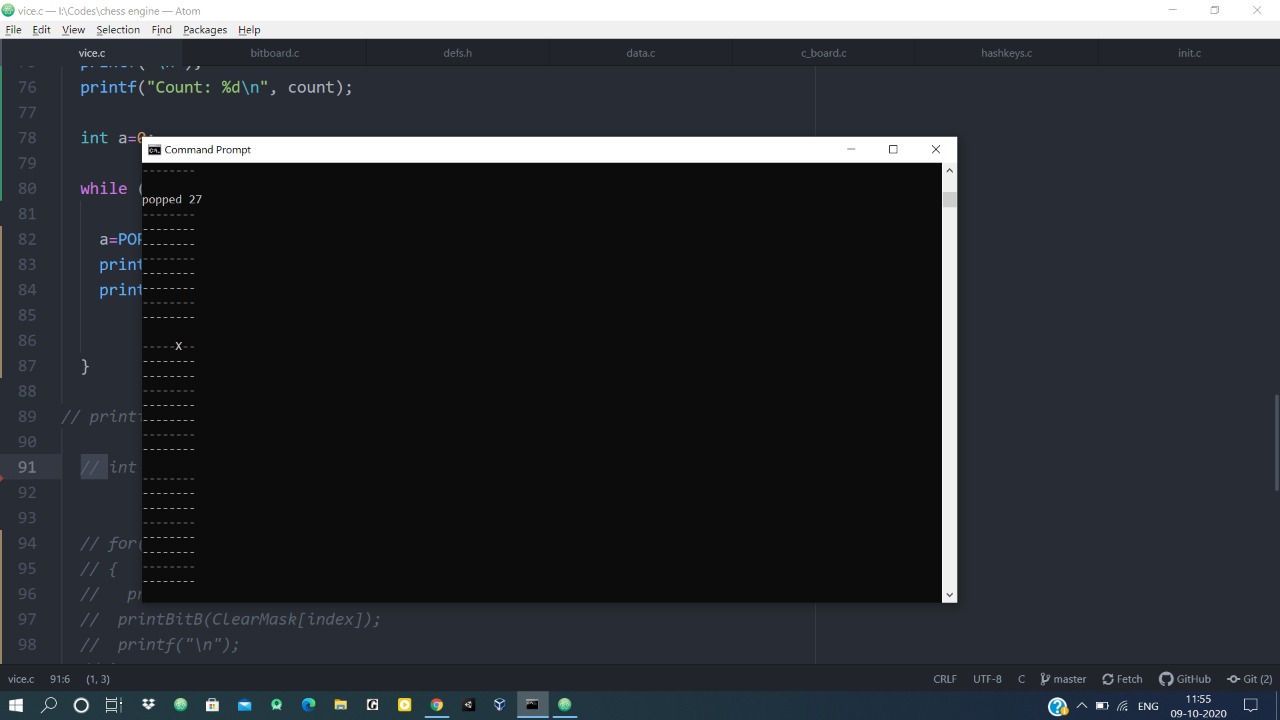
**Algorithm:**

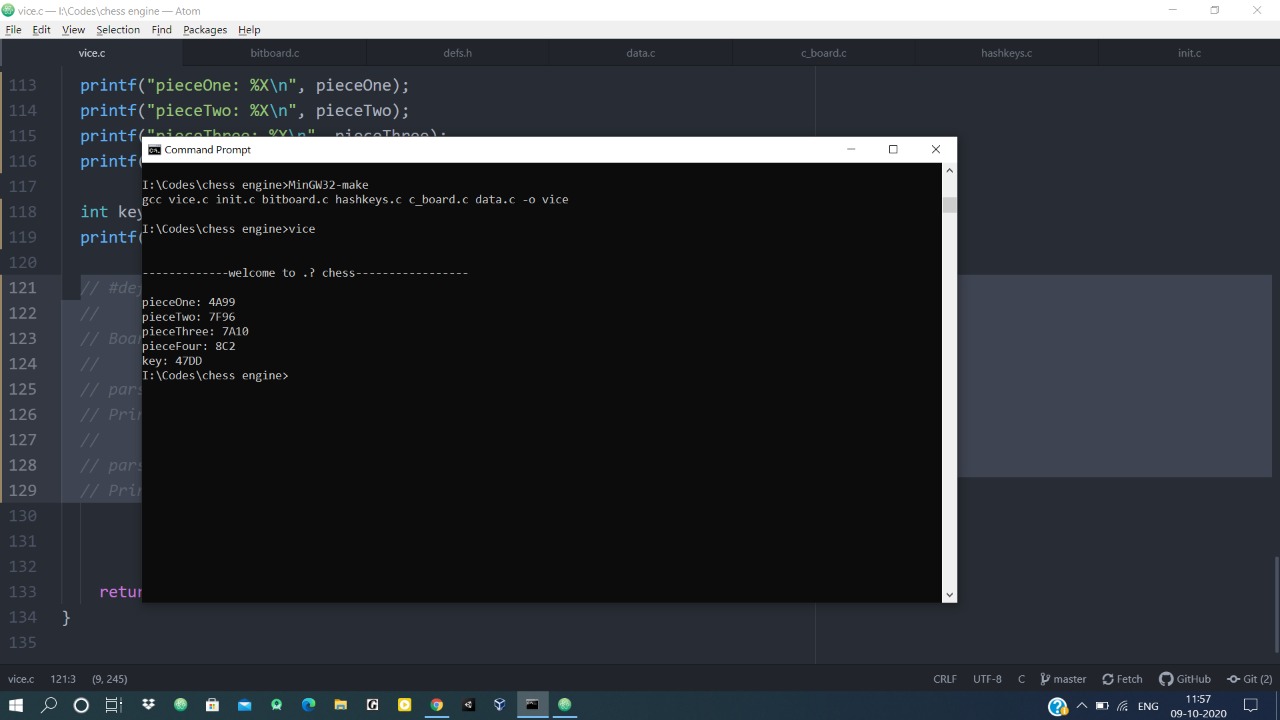
* START
* Enter the name of each player.
* 8 x 8 board will be displayed.
* Game will start with the white pieces
* Players will be able to move their pieces, only to the position which is not pre-occupied by their own pieces.
* Each side will get their turns alternatively.
* Each piece will move according to their predefined rule.
* The last step of each player will be temporarily stored and will change accordingly.
* Any of these player can undo their last move.
* Game will continue until there is any checkmate or a draw situation.
* Name of the winner will be displayed.
* End.

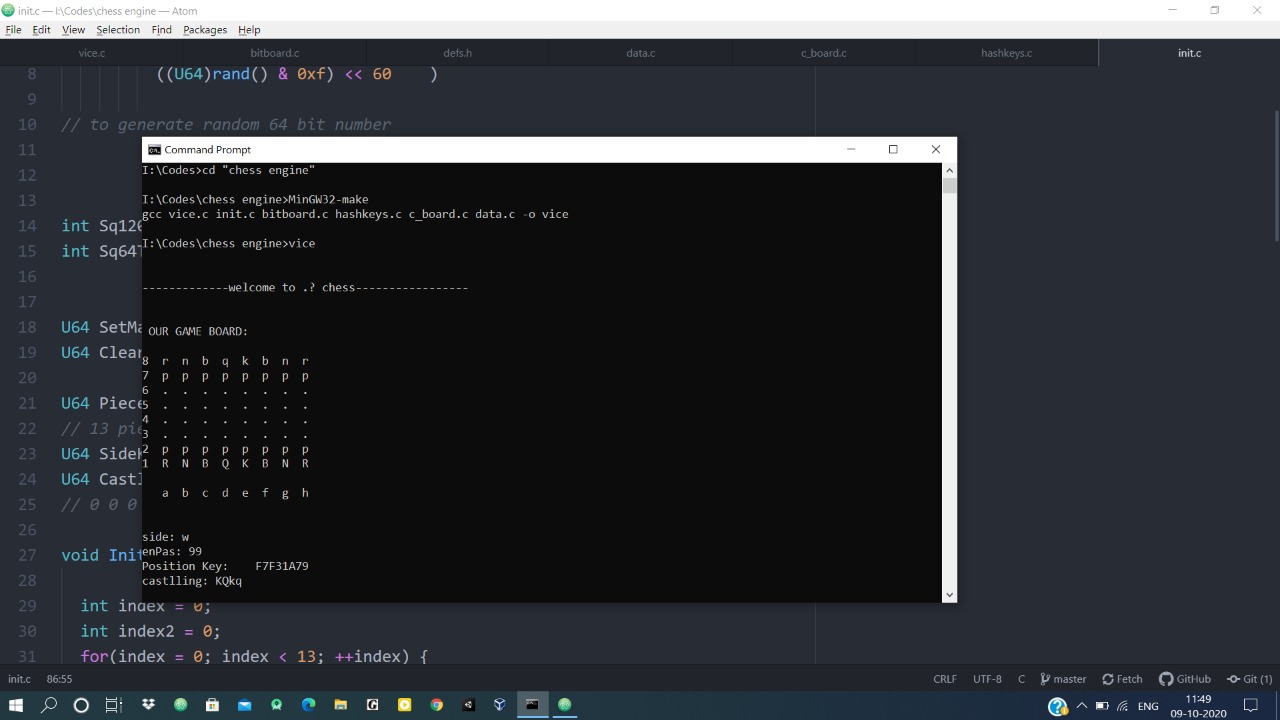
**Code Outputs:**

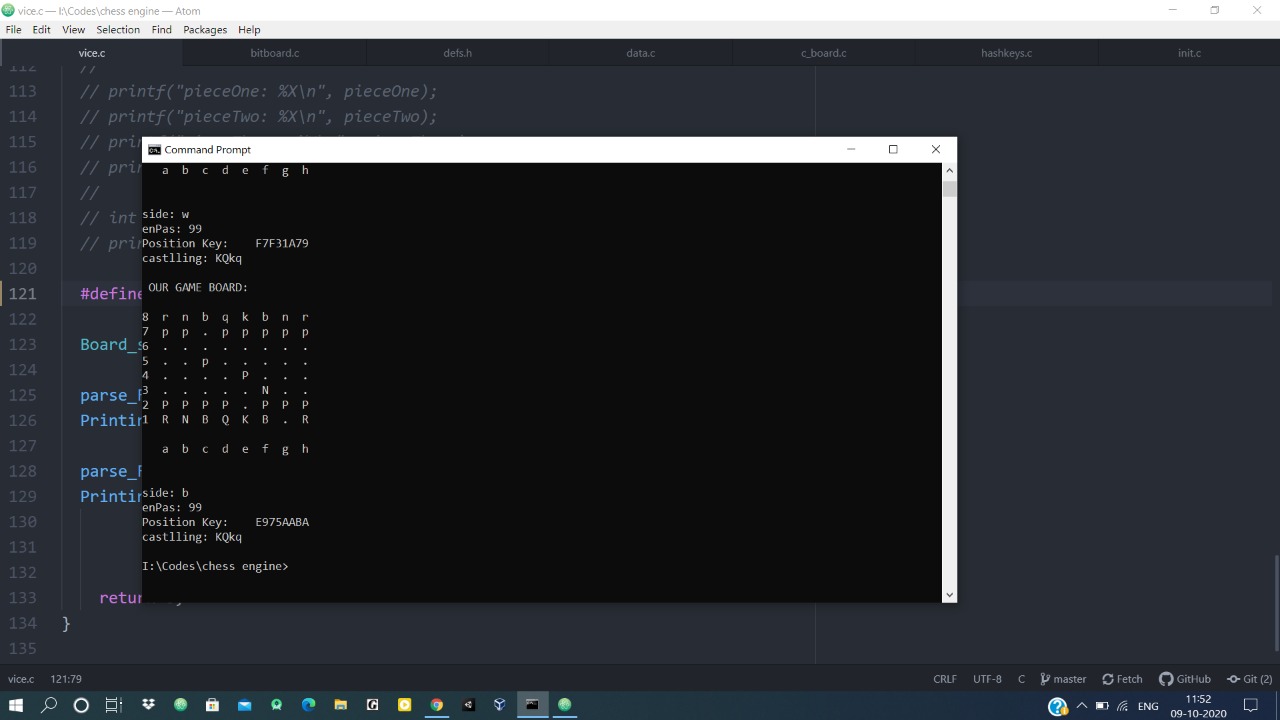
****

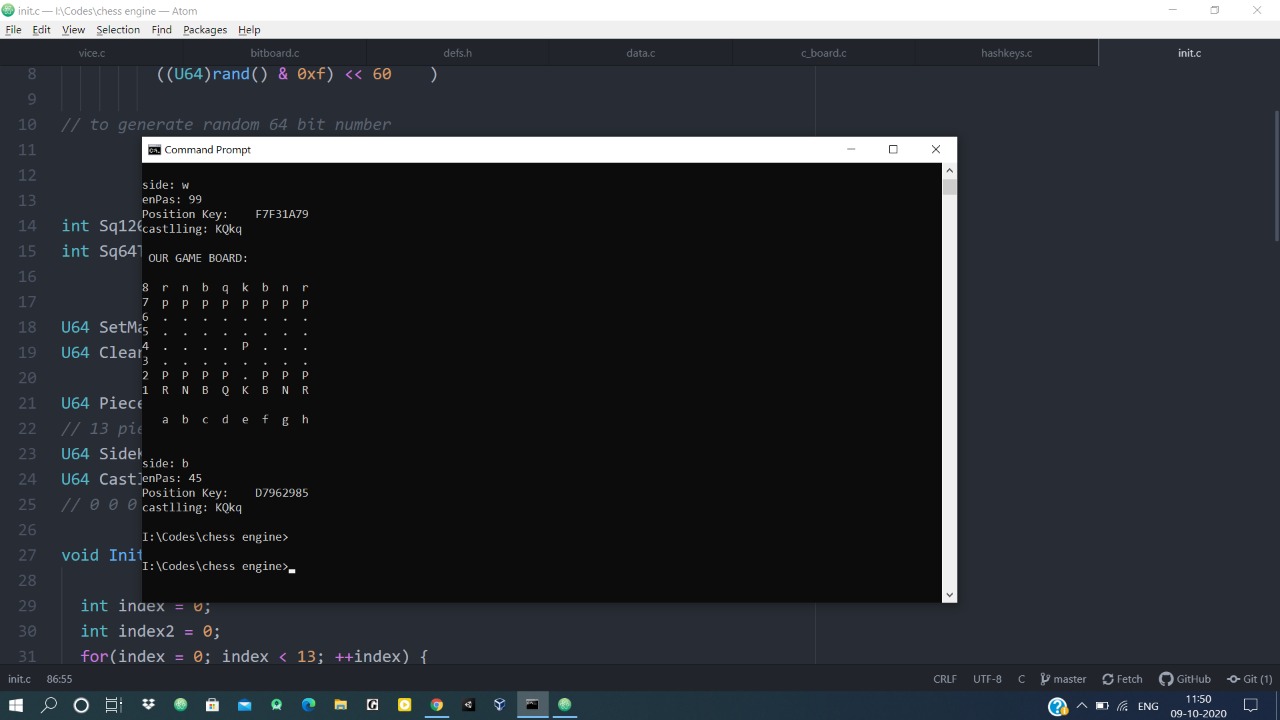
****

****

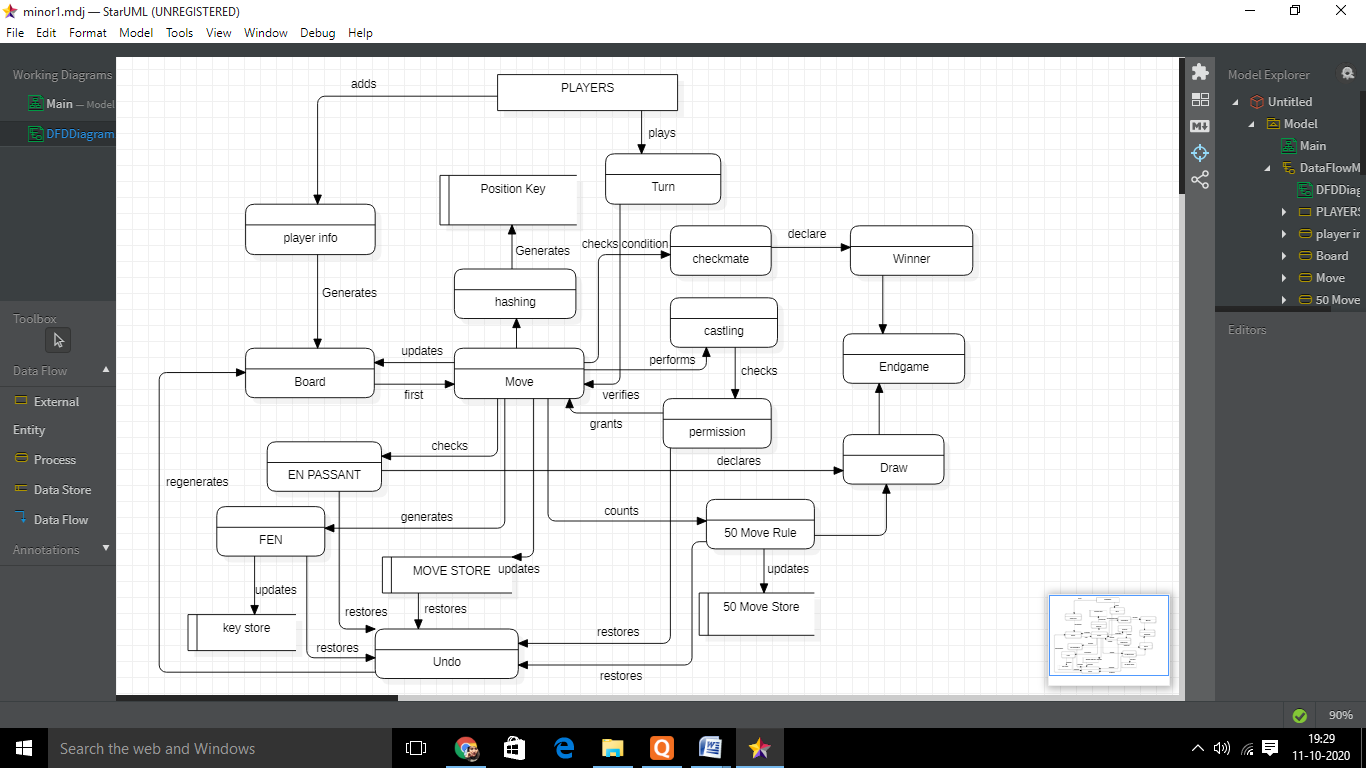
****

****

****

****

**Data Flow Diagram:**

****

**REFERENCES:**

[1]**Essential elements of chess available at** <https://iopscience.iop.org/article/10.1088/1742-6596/1195/1/012013/pdf>

[2] **En passant rule available at** https://www.chess.com/terms/en-passant

[3] **Fifty move available at** https://www.chess.com/forum/view/general/please-explain-50-moves-rule-of-draw

[4] **Quiescence Search available at** <https://www.chessprogramming.org/Quiescence_Search>

[5] **Null Move Pruning available at** <https://bit.ly/3isrQB6>

[6] **Check Extensions** **available at** <https://www.chessprogramming.org/Check_Extensions>

[7] **Forsyth-Edwards Notation** **and Bitboards available at**

https://www.chess.com/blog/the\_real\_greco/representations-of-chess-fen-pgn-and-bitboards

[8] **Principal variation** **available at** <https://bit.ly/36wn0Ay>

[9] **Transposition Table** **available at** https://www.chessprogramming.org/Transposition\_Table

[10] **History Heuristic** **available at** https://www.chessprogramming.org/History\_Heuristic

[11]Performance Testing **available at** <https://www.chessprogramming.org/Perft> and <https://www.guru99.com/performance-testing.html>

**Approved By:**

(Name & Signature) **(**Name & Signature)

**Project Guide Head of Department**

**DR. MONIT KAPOOR DR. MONIT KAPOOR**