**OPTIMAL MOVES IN CHESS**



**BTech/II Year CSE/IV Semester**

**19CSE212/ Data Structures and Algorithms**

**Case Study Report**

**Department of Computer Science and Engineering**

**Amrita School of Computing, Coimbatore**

**Amrita Vishwa Vidyapeetham**

**2022 -2023 Even Semester**

|  |  |
| --- | --- |
| **ROLL NUMBER** | **NAME** |
| CB.EN.U4CSE21201 | Abineha P |
| CB.EN.U4CSE21262 | Suruthi Lavanya A |

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **S.NO.** | **TITLE** | **PG.NO.** |
| 1 | Introduction | 3 |
| 2 | Hybrid Data Structure Implementation | 5 |
| 3 | Practical Applications | 10 |
| 4 | Time and Space Complexity Analysis | 12 |
| 5 | Experimental Evaluation | 13 |
| 6 | Conclusion | 18 |

1. **INTRODUCTION**

Hybrid data structures combine the characteristics and features of multiple traditional data structures to create more efficient and specialized structures for solving specific problems. These data structures leverage the strengths of different types of structures to provide improved efficiency, performance and functionality. Example: Hashed array tree

Chess is a two-player strategy game played on a square board divided into 64 squares of alternating black and white tiles. Each player begins the game with 16 pieces: one king, one queen, two rooks, two knights, two bishops, and eight pawns. The mission of the game is to checkmate your opponent's king, which means putting the king in a position where it is under attack and cannot escape capture. However, the game could end in a stalemate where there is no possibility of any player winning and the game state is forced into a draw. The application of hybrid data structures and optimal moves in chess involves leveraging the benefits of both concepts to enhance the playing experience and improve performance in the game of chess. Hybrid data structures provide efficient storage and retrieval of game states, while optimal moves guide players towards making the best decisions in each situation.

**Objective:**

The objective of integrating hybrid data structures and optimal moves in a chess application is to enhance gameplay, increase the accuracy of move suggestions, and provide efficient analysis and storage of game states. By combining the strengths of hybrid data structures and optimal move algorithms, the application aims to offer players a more enjoyable and competitive chess experience.

Benefits of Hybrid Data Structures and Optimal Moves in Chess Application:

1. Efficient Game State Storage
2. Enhanced Move Generation
3. Performance Optimization
4. Learning and Analysis Tools
5. Strategic Guidance
6. Personalized Recommendations

Three objectives of code implemented:

1. **Simulating Chess openings:**

The Italian Game, Sicilian Defense, The French Defense, The Ruy-Lopez, The Slav Defense. Simulation involving turn wise moves to give player a demo to learn the best and efficient openings that could optimize their moves and increase win rate during gameplay.

1. **Chess game involving ai:**

Implementing the chess game involving player vs AI. AI moves are decided by implementing the NegaMax algorithm and adding alpha beta pruning. The random moves of each piece satisfying valid moves are made more accurate by utilizing weighted graphs.

1. **Displaying graph for each piece in chess:**

The valid moves for each piece are calculated and positions are described as vertices of a graph where the edge of the graph displays the move transition from one tile to another tile. The Knight’s tour problem coded and transitions to be displayed as a graph.

1. **HYBRID DATA STRUCTURE IMPLEMENTATION**

* **Simulating Chess openings:**

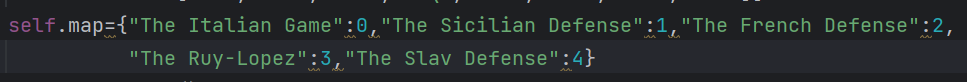
**Aim:**

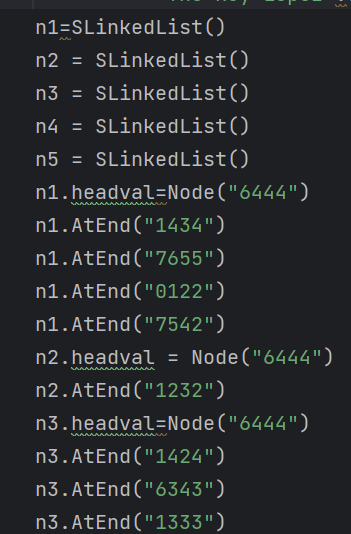
On user’s input of desired Chess opening, the first few moves of the opening are simulated sequentially involving turn based gameplay between white and black pieces.

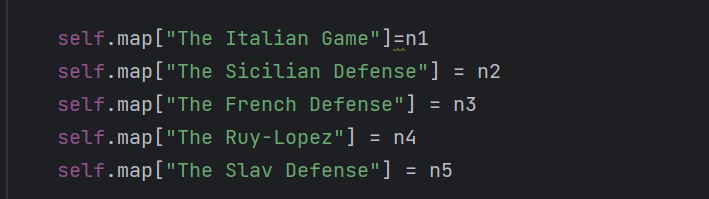
**Hybrid data structure integration:**

HashMap + Singly Linked List

HashMap indexes are the name of the Chess openings which the user enters. Using the index, the value is retrieved which is a singly linked list with ordered moves.







**Design Choice:**

Linked HashMap uses a hybrid data structure to maintain the order of entries in which they were inserted according to specific indexes. Easy way to maintain sequential order of moves to be displayed on the board one by one.

**Efficiency:**

HashMap = O (1): In selecting the index for the corresponding chess opening.

SLL = O (n): n is the number of elements in SLL. Implemented case n<=6, since n is small could be considered constant O (1). To simulate entire chess game starting with a specific opening would be O(n)

* **Chess game involving ai:**

**Aim:**

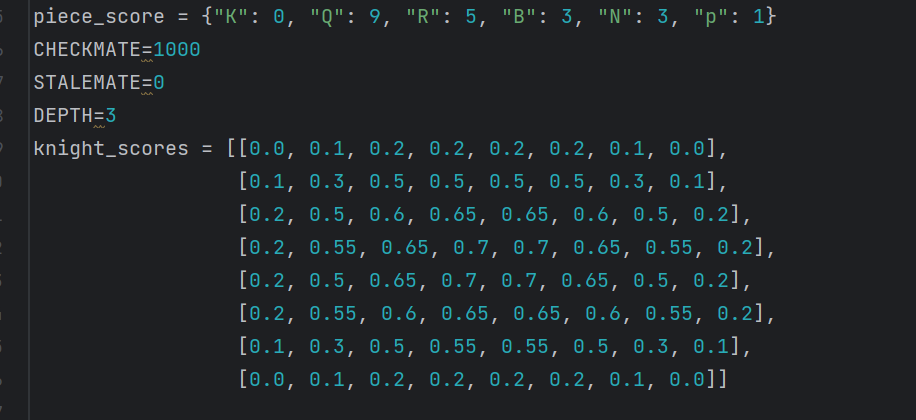
Implementing Real-Time Chess between user vs AI. Board state stored as graph and optimal moves decided using specific algorithms for graphs.

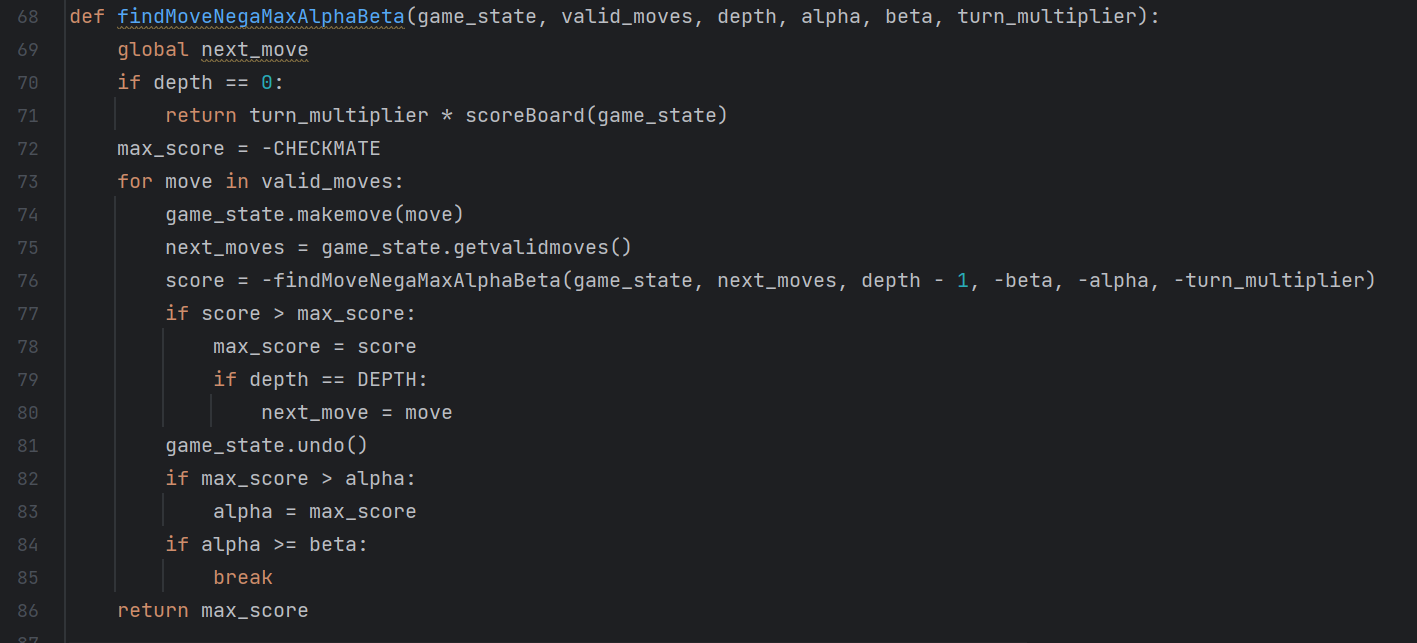
**Hybrid data structure integration:**

Graph + Weighted Graph + Search Trees

Using an unweighted graph to represent 8x8 chess board. Generating all valid possible moves for each user and AI’s turn to validate the correctness and possibility of making the move. AI using weighted graphs to learn which move is the best among all possible valid moves. Implementation of NegaMax with alpha-beta pruning for search game tree.







**Design Choice:**

Since, the 8x8 board is of the same representation as a 2-D matrix, placing each piece on the board would be easy as a graph. The position of each piece or tile could be calculated as considering a graph spread out across the coordinate axes. Weighted graph provides the AI with precise vertices on which the piece would have high score making it the best possible move. Game search trees are important in artificial intelligence because one way to pick the best move in a game is to search the game tree which has a record of all future gameboard states and utilizes numerous tree search algorithms, combined with minimax-like rules to prune the tree.

**Efficiency:**

Board = O(1) : In selecting moving a chess piece form one tile to another tile.

Scores = O(n^2) : nxn to traverse throughout the board and generate all possible moves. The time complexity of the Minimax algorithm is O(bd), where b is the branching factor (number of branches for each node) and d is the depth of the tree.

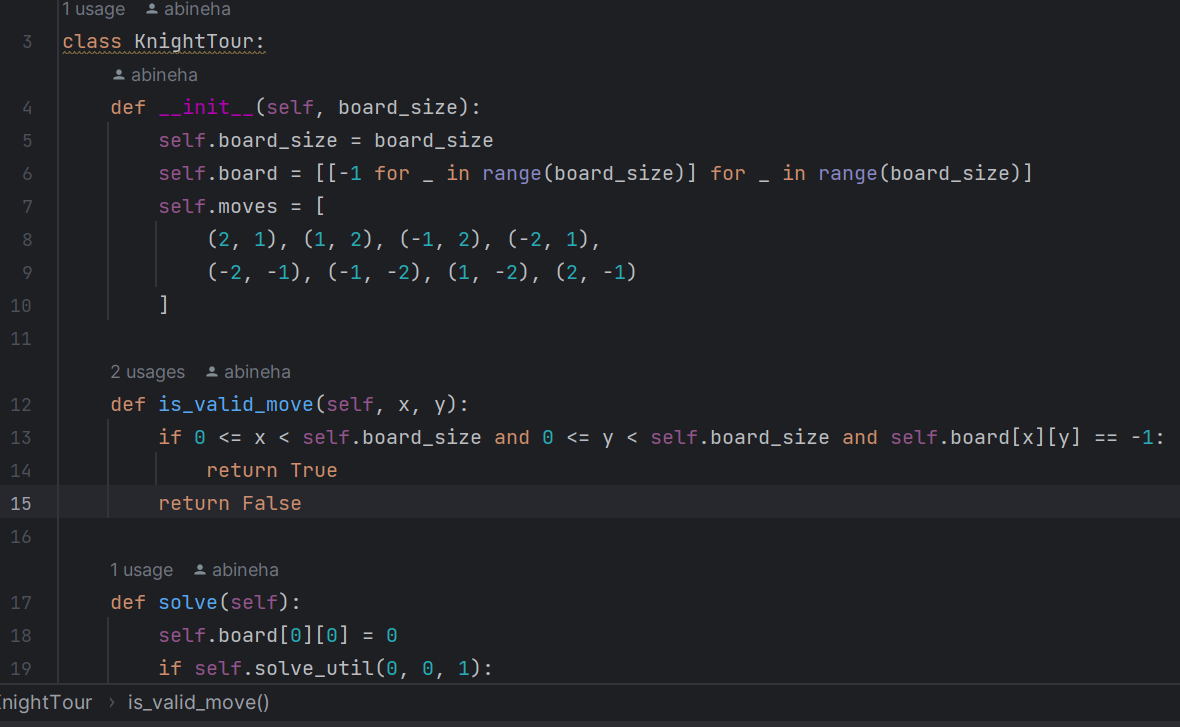
* **Displaying graph for each piece in chess:**

**Aim:**

Graphical representation of moves of each unique piece which shows behavior of each piece.

**Hybrid data structure integration:**

Graph and array of tuples.





**Design Choice:**



**Efficiency:**

O(n\*n): to traverse throughout 8x8 board and generate all possible moves to plot graph using matplotlib.

**GitHub Repository:** https://github.com/abineha/Chess-optimal-moves-simulation

1. **PRACTICAL APPLICATIONS**

**1. Linked Hash Map:**

The main advantage of using LinkedHashMap is that it maintains and tracks the order of insertion where elements can be inserted and accessed in their order. It maintains a linked list of the entries in the map, in the order in which they were inserted. This allows insertion-order iteration over the map. If one inserts the key again into the LinkedHashMap, the original order is retained.

* Shopping cart (Cart No: Vs Items chosen)
* Cache implementation (Preserves insertion order and allows efficient removal)
* LRU (Least Recently Used)
* Browser History (URL as entry in map)
* Job Scheduling (Maintains order and allows quick access)

**2. Graphs:**

Graphs are effective tools in representing complex data. They can help to uncover patterns, trends, and insights that may be difficult to see using other methods. Graphs can be processed efficiently using graph algorithms, which are specifically designed to work with graph data structures. This makes it possible to perform complex operations on large datasets quickly and effectively.

* Social Networking (model and analyze connections between users)
* Recommendation systems (items as nodes and captures relationships as edges)
* Supply chain systems (locations as nodes, transportation as routes)
* Computer networks (devices as nodes and connections as edges)
* Biological systems (protein-protein interaction networks, gene regulatory networks, or metabolic pathways)

1. **Search Trees**

A search tree is a tree data structure used for locating specific keys from within a set. In order for a tree to function as a search tree, the key for each node must be greater than any keys in subtrees on the left, and less than any keys in subtrees on the right.

* Databases (B-trees, AVL trees, are used in database management systems)
* File systems (efficient searching and navigation through the file system)
* Autocomplete and spell checks (store a dictionary of words, allowing for efficient prefix-based searches to suggest completions or identify potential spelling errors)
* NLP (store n-grams, linguistic patterns, or language models)
* Game AI and Decision Trees (decision trees for making strategic choices)

**4. Arrays**

* Data storage and retrieval
* Mathematical and statistical computations
* Image and signal processing
* Matrices and linear algebra
* Performance optimization

**5. Optimization of chess moves**

The logic behind implementation could be used for various strategical and tactical boardgames and real-time optimization problems.

* Tactical Awareness
* Calculation and Visualization
* Endgame Knowledge
* Continuous learning
* Time Management

1. **TIME AND SPACE COMPLEXITYN ANALYSIS**

* **Simulating Chess openings:**

HashMap = O (1) (time complexity): In selecting the index for the corresponding chess opening.

O(n) (space complexity): for n positions in table

SLL = O (n) (time complexity): n is the number of elements in SLL. Implemented case n<=6, since n is small could be considered constant O (1). To simulate entire chess game starting with a specific opening would be O(n).

O(n) (space complexity): for n entries in list

* **Chess game involving ai:**

The best-case time complexity of alpha-beta pruning is O(bd/2) or O(bd) (where b is the branching factor and d is the depth of the game tree), which is significantly lesser than the minimax algorithm, which has a time complexity of O(bd).

Board = O (1) (time complexity): In selecting moving a chess piece form one tile to another tile.

O(n^2) (space complexity): 2d array each array of O(n)

Scores = O(n^2) (time complexity): (n x n) to traverse throughout the board and generate all possible moves. The time complexity of the Minimax algorithm is O(bd), where b is the branching factor (number of branches for each node) and d is the depth of the tree.

* **Displaying graph for each piece in chess:**

O(n\*n) - (time complexity): for each possible position

O(n\*n) - (time complexity): for knight’s tour

O(n^2) (space complexity): 2d array each array of O(n)

* **Negamax with Alpha-Beta Pruning:**

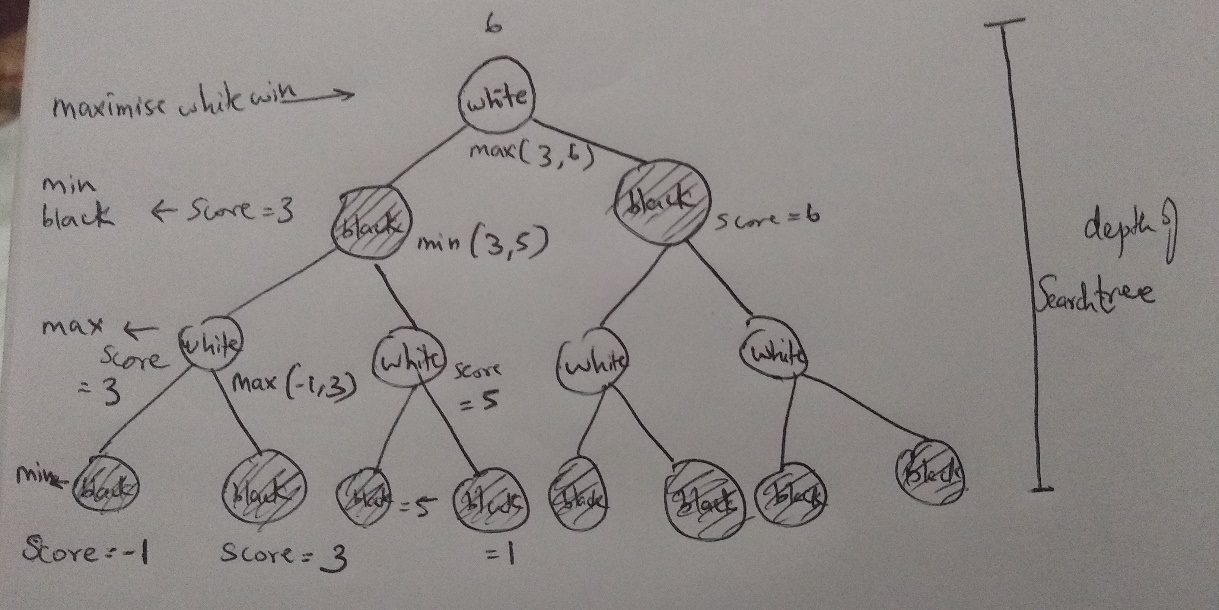
**Mini-max** algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory. It provides an optimal move for the player assuming that opponent is also playing optimally. Mini-Max algorithm uses recursion to search through the game-tree. Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit.

The minimax algorithm performs a depth-first search algorithm for the exploration of the complete game tree. The minimax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.

**Negamax** is a more efficient version of Minimax for games where the scoring is zero-sum. **Zero-sum** is a situation, in game theory, in which one person's gain is equivalent to another's loss, so the net change in benefit is zero. [Alpha-beta pruning](https://en.wikipedia.org/wiki/Alpha-beta_pruning) can decrease the number of nodes the negamax algorithm evaluates in a search tree.

The need for pruning came from the fact that in some cases decision trees become very complex. In that tree, some useless branches increase the complexity of the model. So, to avoid this, Alpha-Beta pruning comes to play so that the computer does not have to look at the entire tree.

These search algorithms are essential in turn-based games. It allows the program to look ahead at possible future positions before deciding what move it wants to make in the current position.



**Mini-Max Search Tree**

**Pseudocode:**

Function minimax(position,depth,alpha,beta,maximizingplayer)

If depth == 0 or game over in position

return static evaluation of position

If maximisingplayer

maxEval = -infinity

for each child of position

eval=minimax (child, depth-1, alpha, beta, false)

maxEval=max(maxEval,eval)

alpha=max(alpha,eval)

if beta<=alpha

break

return maxEval

else:

miniEval = +infinity

for each child of position

eval=minimax (child, depth-1, apha, beta. true)

minEval=min(minEval,eval)

beta=min(beta,eval)

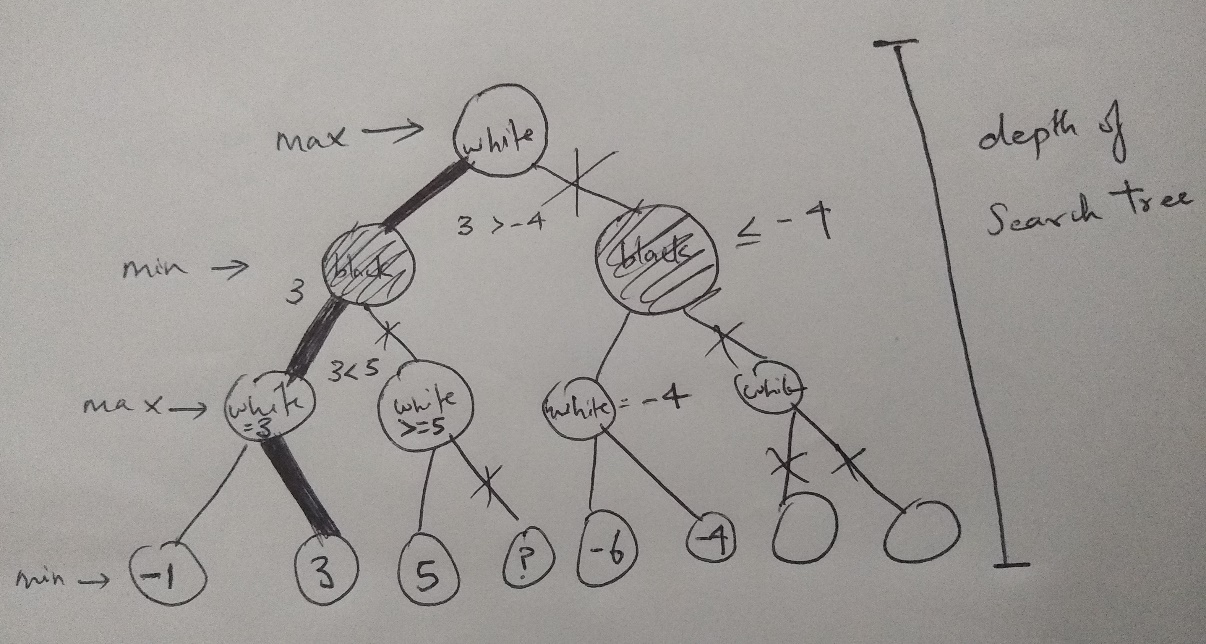
if beta<=alpha

break

return miniEval

//initial call

Minimax (currentPosition,3, -infinity, +infinity, true)



**Mini-Max Search Tree with alpha-beta Pruning**

1. **EXPERIMENTAL EVALUATION AND RESULTS**

* **Simulating Chess openings:**

****

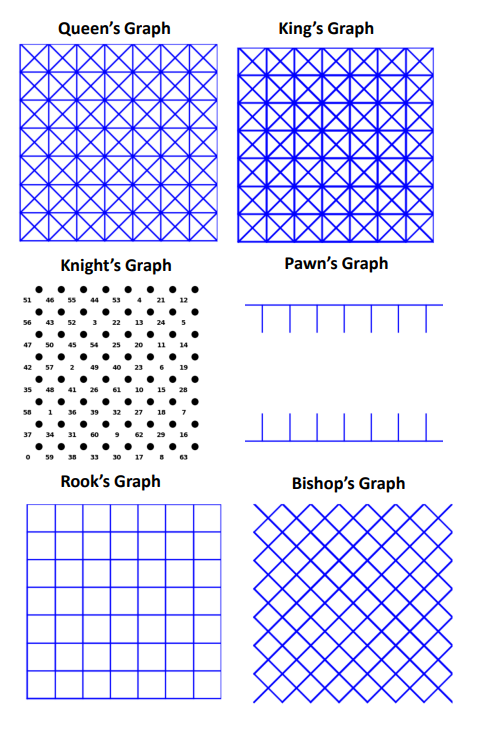
**The Italian Game Opening**

* **Chess game involving ai:**

****

**Player vs AI (White vs Black)**

* **Displaying the graph for each piece in chess:**

****

**Matplotlib generated graphs**

1. **CONCLUSION**

We have implemented chess, optimized possible chess moves and evaluated the score for each status of the gameboard. The utilized hybrid data structures helped in having a clear representation and depiction of complex game states. The hybrid data structures efficiently stored data making it easy to track past moves and store future valid moves.

Trade-offs had to be done in the field of utilizing space. The space complexity should be further reduced as our code have multiple graphs for each chess piece making it not the best in terms of space and memory utilization.

The logic used in our code could be further applied in real-time situation requiring the need of complex operations and optimization requirements.

**References:**

1. https://www.researchgate.net/figure/Enhanced-NegaMax-with-Alpha-Beta-Property-Pseudo-Code\_fig4\_262672371
2. https://observablehq.com/@peatroot/chess-graphs
3. https://runestone.academy/ns/books/published/pythonds/Graphs/BuildingtheKnightsTourGraph.html