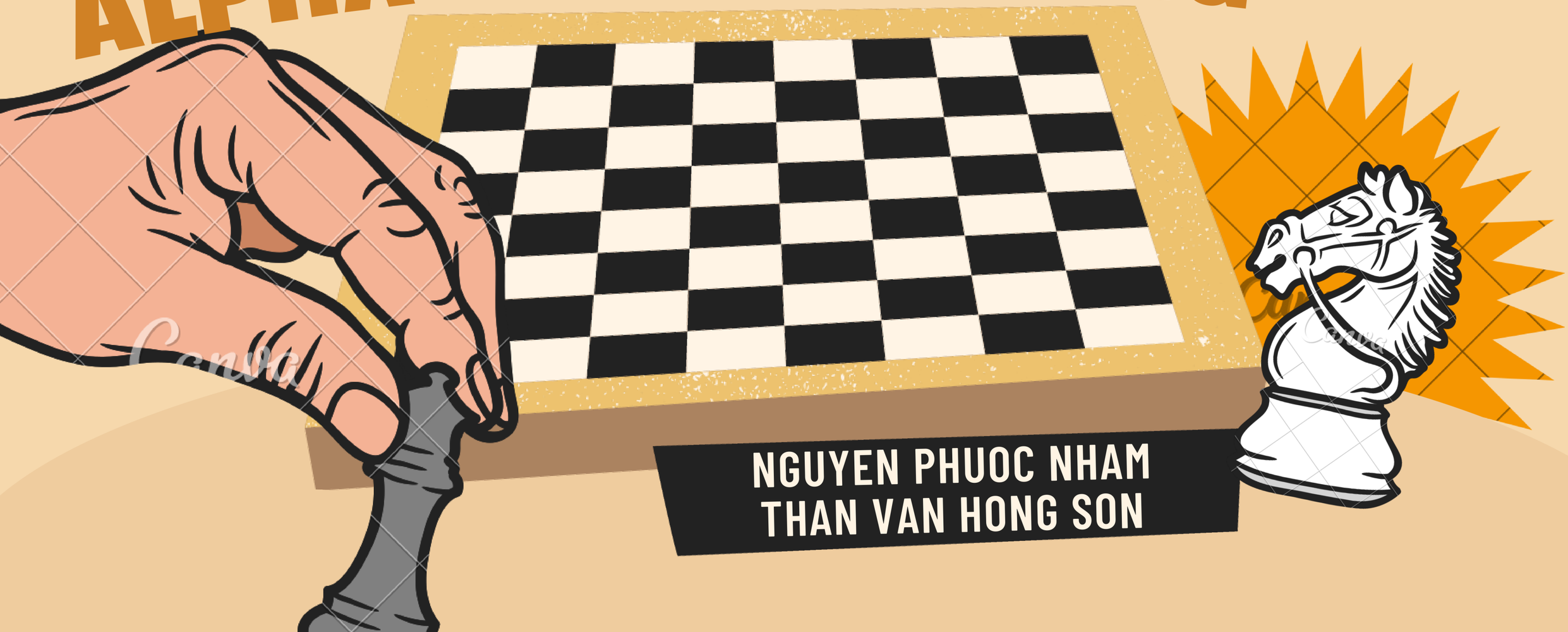
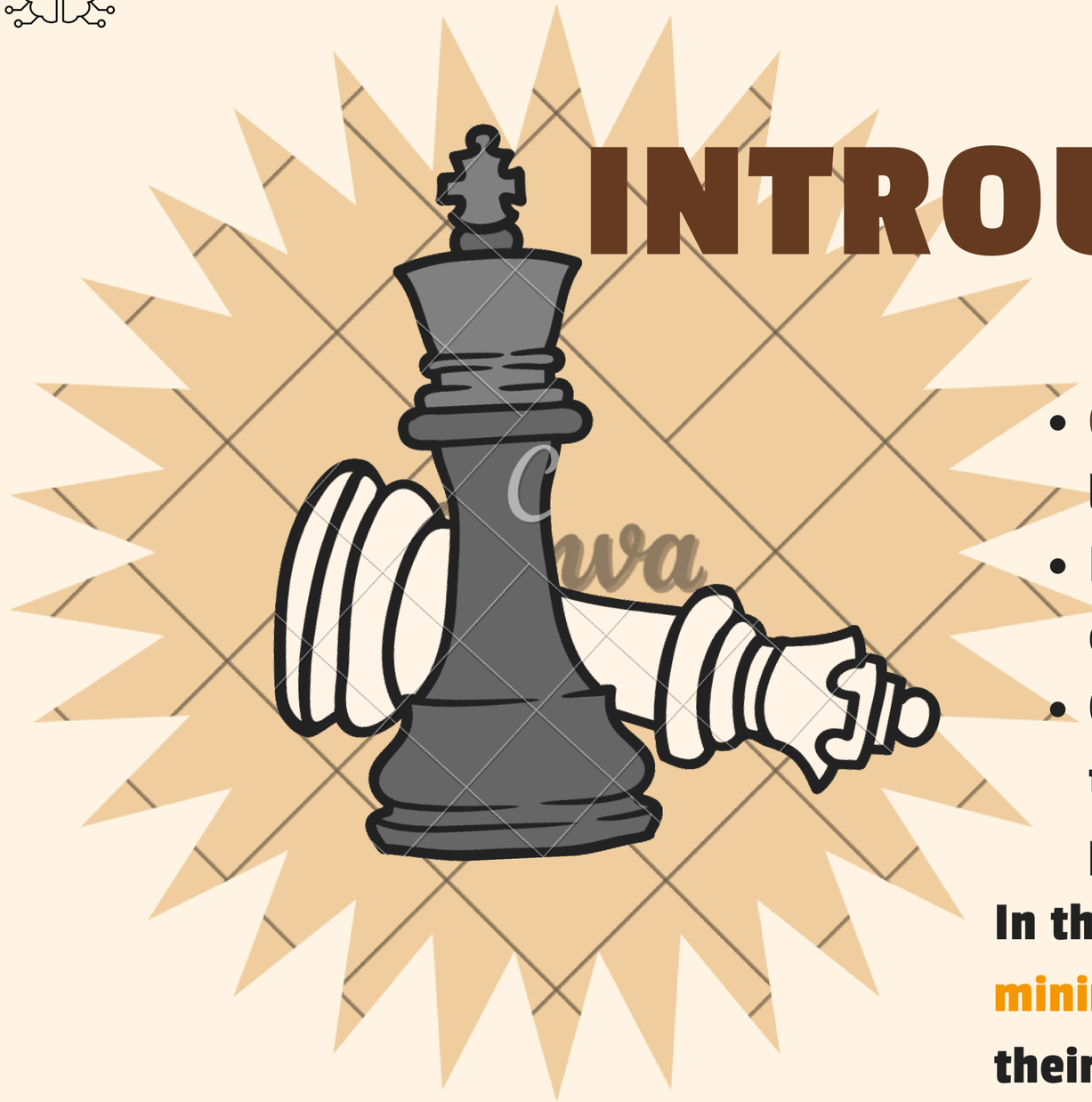
ARTIFICIAL INTELLIGENCE

MINIMAX + ALPHA-BETA PRUNING CHESS



NGUYEN PHUOC NHAM
THAN VAN HONG SON



INTROUCTION - CHESS

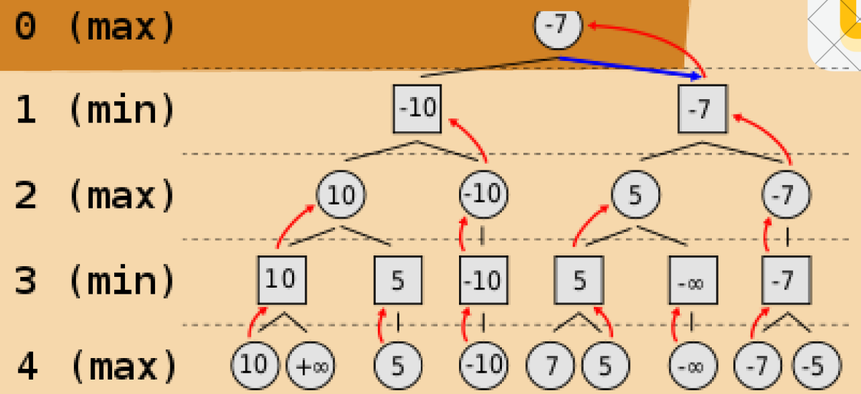
- **Chess** is a popular board game that requires a high level of strategic **thinking** and **planning**.
- It is an excellent game to showcase the capabilities of artificial intelligence (AI) systems.
- One approach to creating a chess-playing AI is to use the minimax algorithm, along with alpha-beta pruning.

In this report, we will explore the basic concepts of the **minimax algorithm** and **alpha-beta pruning**, as well as their implementation in a chess-playing AI.

ARTIFICIAL INTELLIGENCE

MINIMAX ALGORITHM

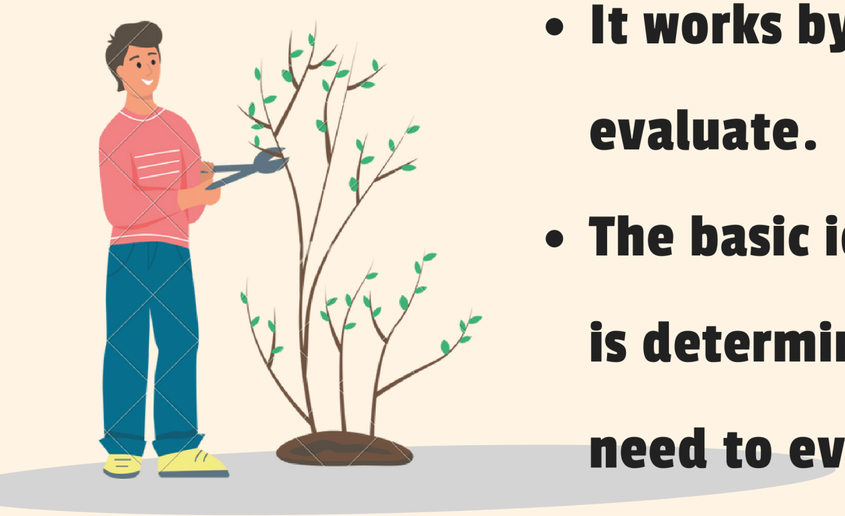
- The minimax algorithm is a decision-making algorithm used in two-player games such as chess.
- It works by considering all possible moves that the player can make, and then analyzing the resulting board states to determine the best move to make.
- The algorithm assumes that the opponent will make the move that is worst for the player, and the player will make the move that is best for them. It then chooses the move that results in the best outcome for the player, assuming the worst-case scenario.



- The minimax algorithm is recursive, meaning that it calls itself to evaluate each possible move.
- At the lowest level of the recursion, the algorithm evaluates the board state and assigns it a score.
- This score is then passed up the tree to the next level of recursion, where the algorithm chooses the best move based on the scores assigned to each possible move.

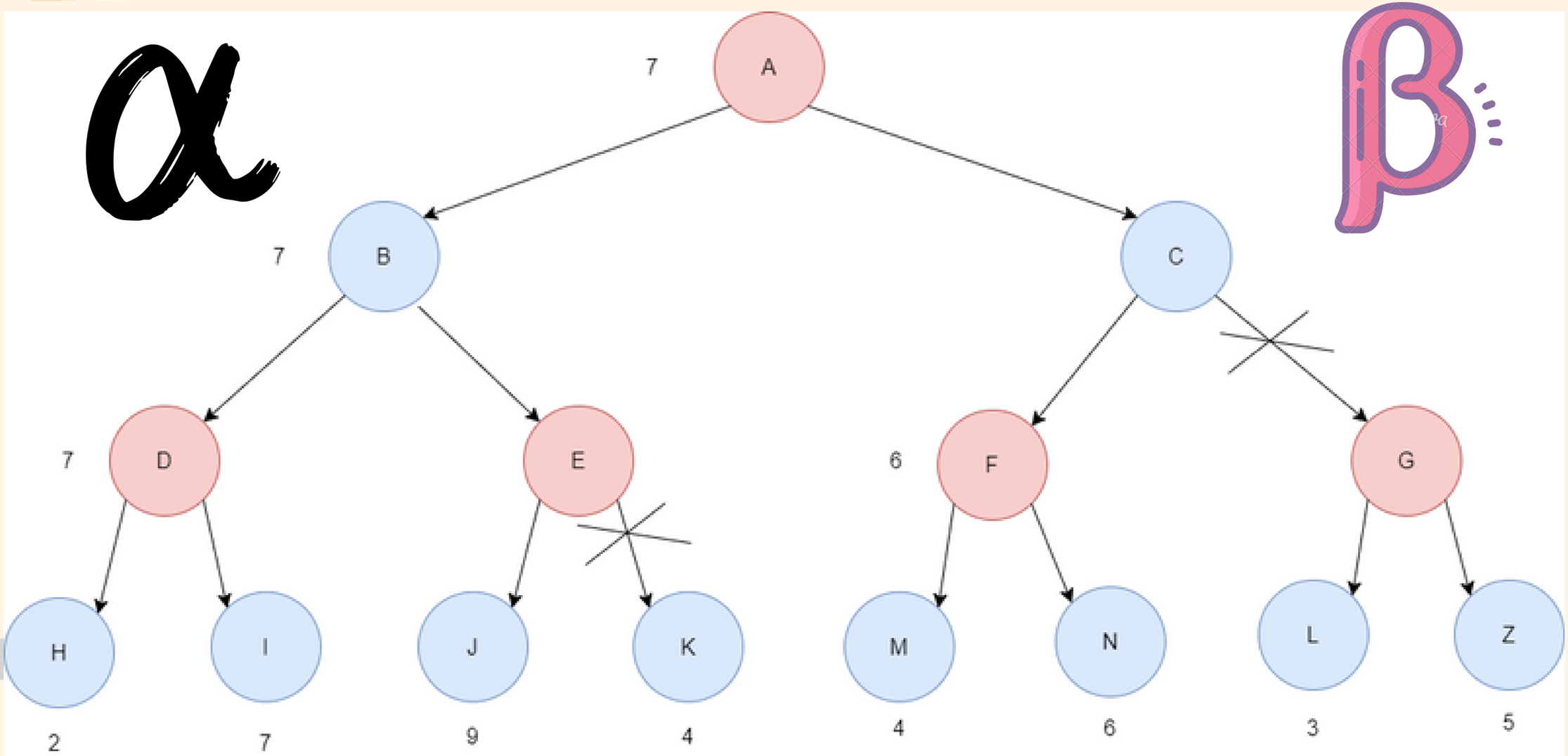
ALPHA-BETA PRUNING

- **Alpha-beta pruning** is a technique used to **improve** the performance of the **minimax** algorithm.
- It works by **reducing** the number of nodes that the algorithm needs to evaluate.
- The basic idea behind alpha-beta pruning is that if a node in the minimax tree is determined to be worse than a previously evaluated node, then there is no need to evaluate any of its children. This is because the player will never choose that node, as there is a **better** option available.



ARTIFICIAL INTELLIGENCE

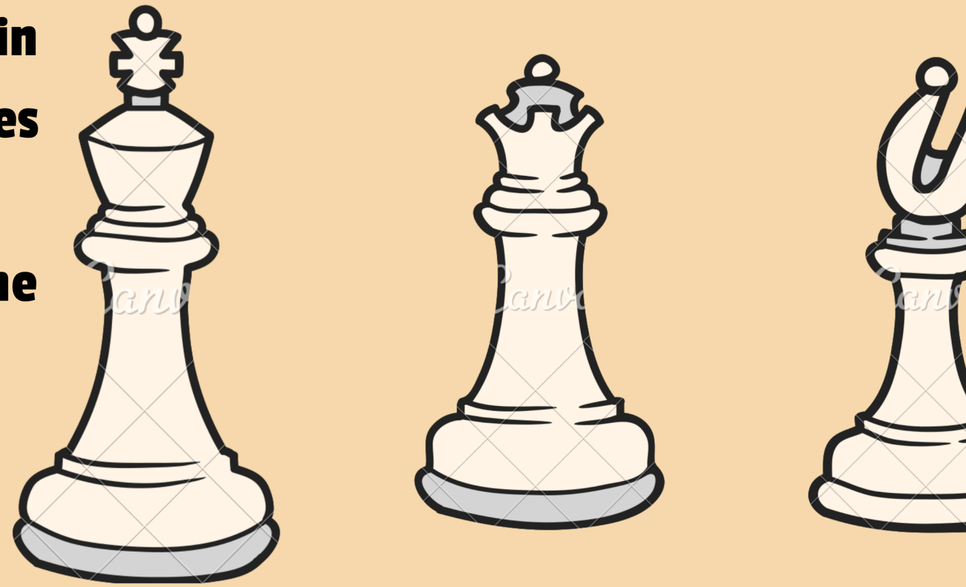
ALPHA-BETA PRUNING



IMPLEMENTING MINIMAX + A-B PRUNING IN CHESS AI

```
76  def reward(board, isLower):
77      totalPoints = 0
78      totalPoints += value(board, isLower) * 10
79      totalPoints += matchStatus(board, isLower) * 3 + \
80          matchStatus(board, not isLower) * 2
81
82      return totalPoints
```

- To implement the minimax algorithm and alpha-beta pruning in a chess AI, we need to define a scoring function that evaluates the board state.
- The scoring function should take into account factors such as the number of pieces on the board, their position, and their value.

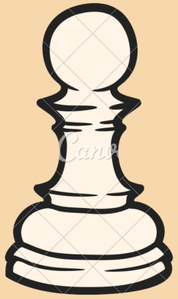


ARTIFICIAL INTELLIGENCE

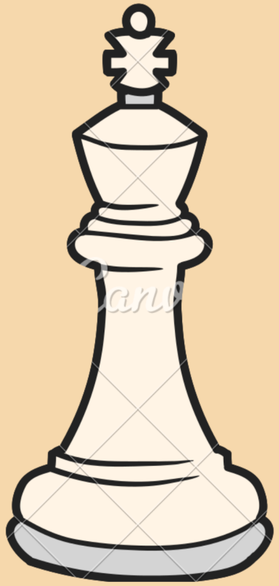
IMPLEMENTING MINIMAX + A-B PRUNING IN CHESS AI

```
6     pawnUpper = np.array([[0.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0],
7                           [5.0,  5.0,  5.0,  5.0,  5.0,  5.0,  5.0,  5.0],
8                           [1.0,  1.0,  2.0,  3.0,  3.0,  2.0,  1.0,  1.0],
9                           [0.5,  0.5,  1.0,  2.5,  2.5,  1.0,  0.5,  0.5],
10                          [0.0,  0.0,  0.0,  2.0,  2.0,  0.0,  0.0,  0.0],
11                          [0.5, -0.5, -1.0,  0.0,  0.0, -1.0, -0.5,  0.5],
12                          [0.5,  1.0, 1.0,  -2.0, -2.0,  1.0,  1.0,  0.5],
13                          [0.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0]
14                          ])
15
16    pawnlower = pawnUpper[::-1]
```

Value, position



PAWN



KING

```
62    kingUpper = np.array([
63                          [-3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0],
64                          [-3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0],
65                          [-3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0],
66                          [-3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0],
67                          [-2.0, -3.0, -3.0, -4.0, -4.0, -3.0, -3.0, -2.0],
68                          [-1.0, -2.0, -2.0, -2.0, -2.0, -2.0, -2.0, -1.0],
69                          [2.0,  2.0,  0.0,  0.0,  0.0,  0.0,  2.0,  2.0],
70                          [2.0,  3.0,  1.0,  0.0,  0.0,  1.0,  3.0,  2.0]
71                          ])
72
73    kingLower = kingUpper[::-1]
```

IMPLEMENTING MINIMAX + A-B PRUNING IN CHESS AI

Status

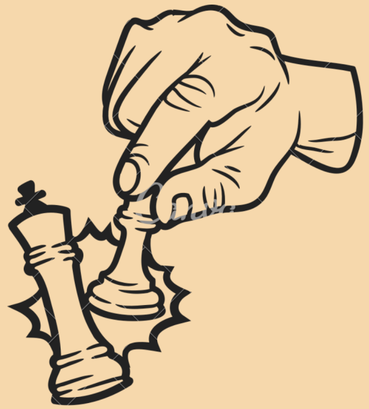
```
127     def matchStatus(board, isLower):
128         """
129         return:
130         0: NORMAL
131         1: STEALMATE
132         2: CHECK
133         inf: CHECKMATE
134         """
```

```
151         if isLower:
152             if (col, row) in enermyList:
153                 if canGosOfKing:
154                     return 2
155                 else:
156                     return 1000
157             if not canGosOfKing:
158                 return 1
159         else:
160             if (col, row) in enermyList:
161                 if canGosOfKing:
162                     return -2
163                 else:
164                     return -1000
165             if not canGosOfKing:
166                 return -1
167
168         return 0
```

If the position is in the enemy's can go list

If the King can go

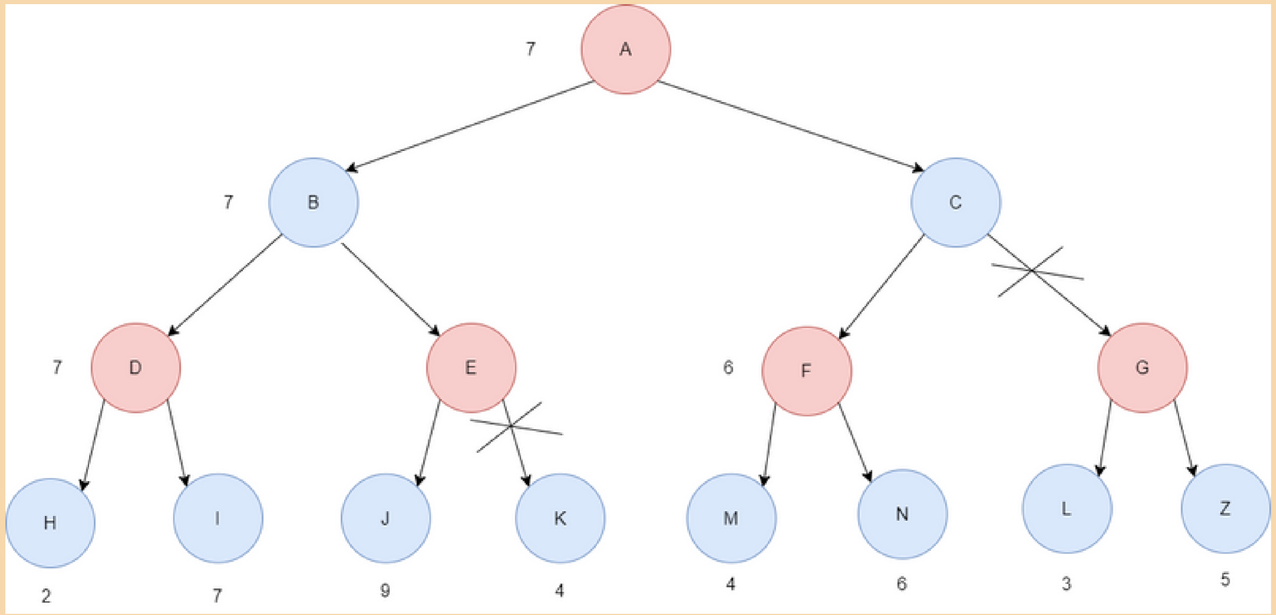
If the King can't go



CHECKMATE

IMPLEMENTING MINIMAX + A-B PRUNING IN CHESS AI

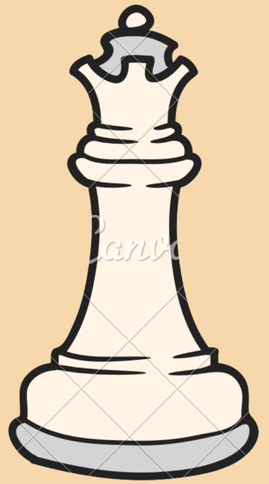
- The AI would start by evaluating all possible moves that it can make, and then recursively evaluate the resulting board states using the minimax algorithm with alpha-beta pruning.
- The algorithm would continue to evaluate each possible move until it reaches a specified depth, at which point it would return the score of the board state.



ARTIFICIAL INTELLIGENCE

IMPLEMENTING MINIMAX + A-B PRUNING IN CHESS AI

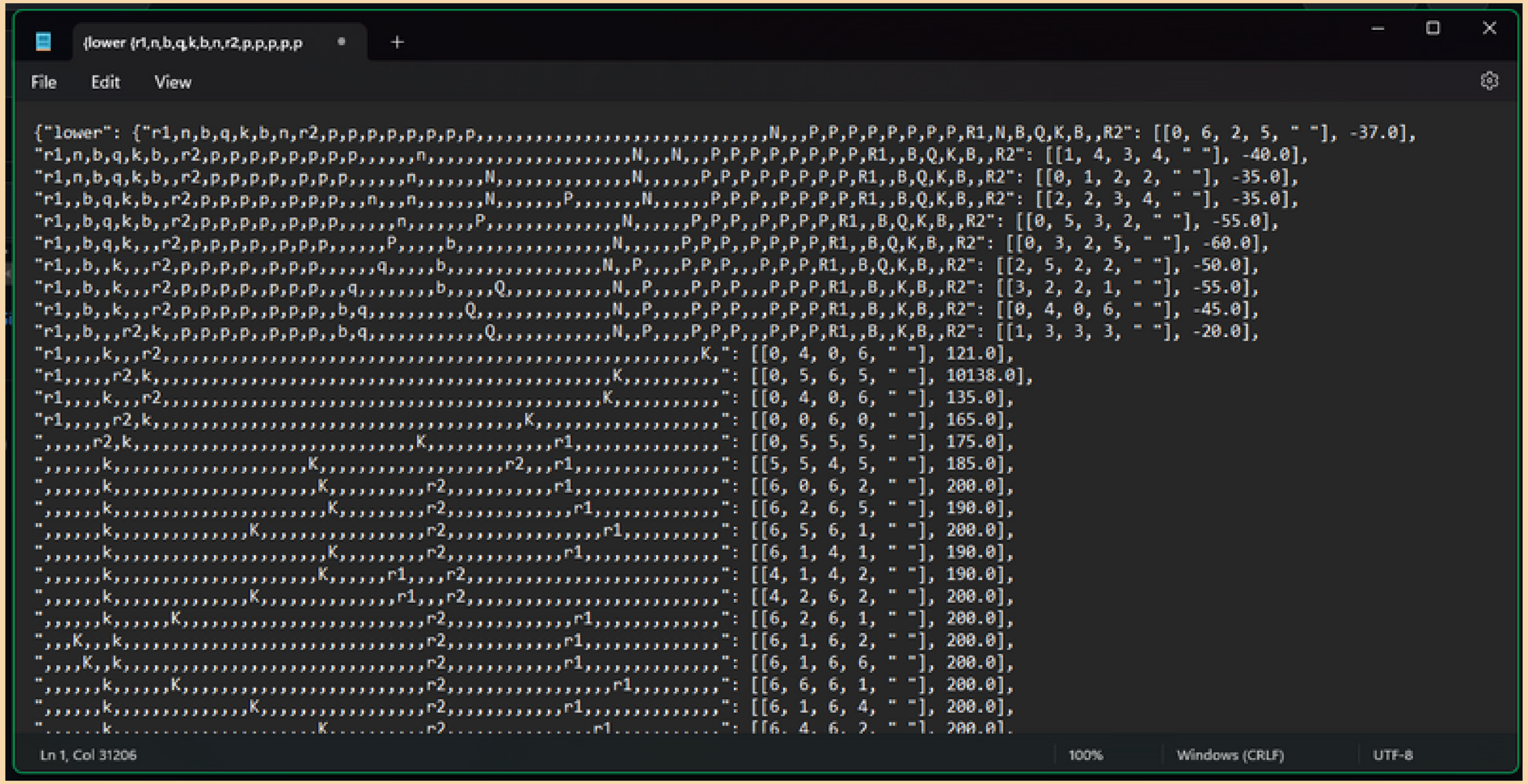
```python
def Minimax(node, depth, Pmax, Pnow,
            isMoved = {
                'k': False, 'K': False, 'r1': False,
                'R1': False, 'r2': False, 'R2': False
                },
            alpha = -inf, beta = inf):
    """
    node là node hiện tại
    depth là độ sâu
    Pmax là player cần tìm Max
    Pnow là player hiện tại
    isMoved dùng để kiểm tra các quân xe, vua có di chuyển hay chưa để nhập thành
    alpha là giá trị tốt nhất của người chơi maximizing (mặc định là -inf)
    beta là giá trị tốt nhất của người chơi minimizing (mặc định là inf)
    """
```

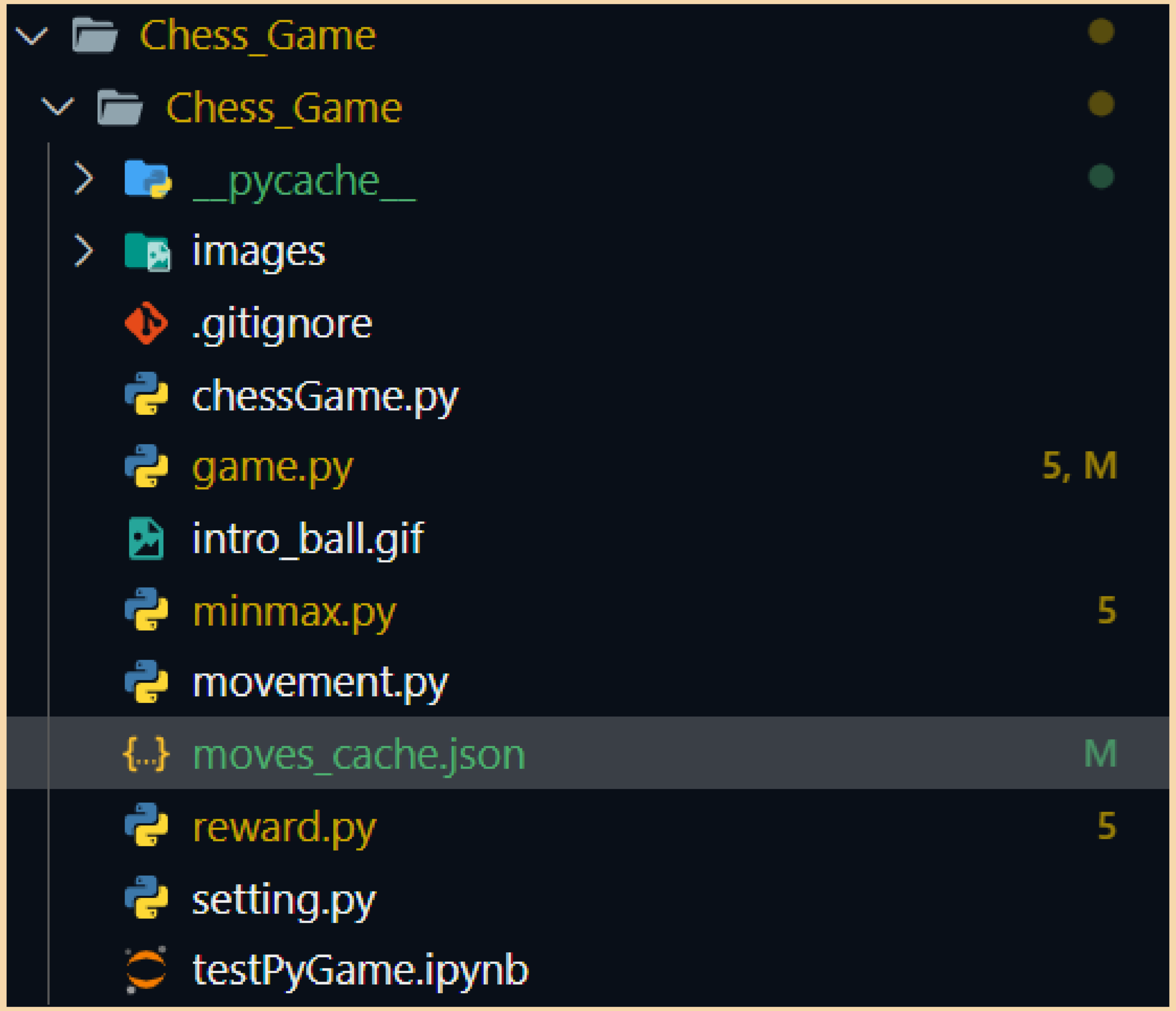


QUEEN

ARTIFICIAL INTELLIGENCE

IMPLEMENTING MINIMAX + A-B PRUNING IN CHESS AI

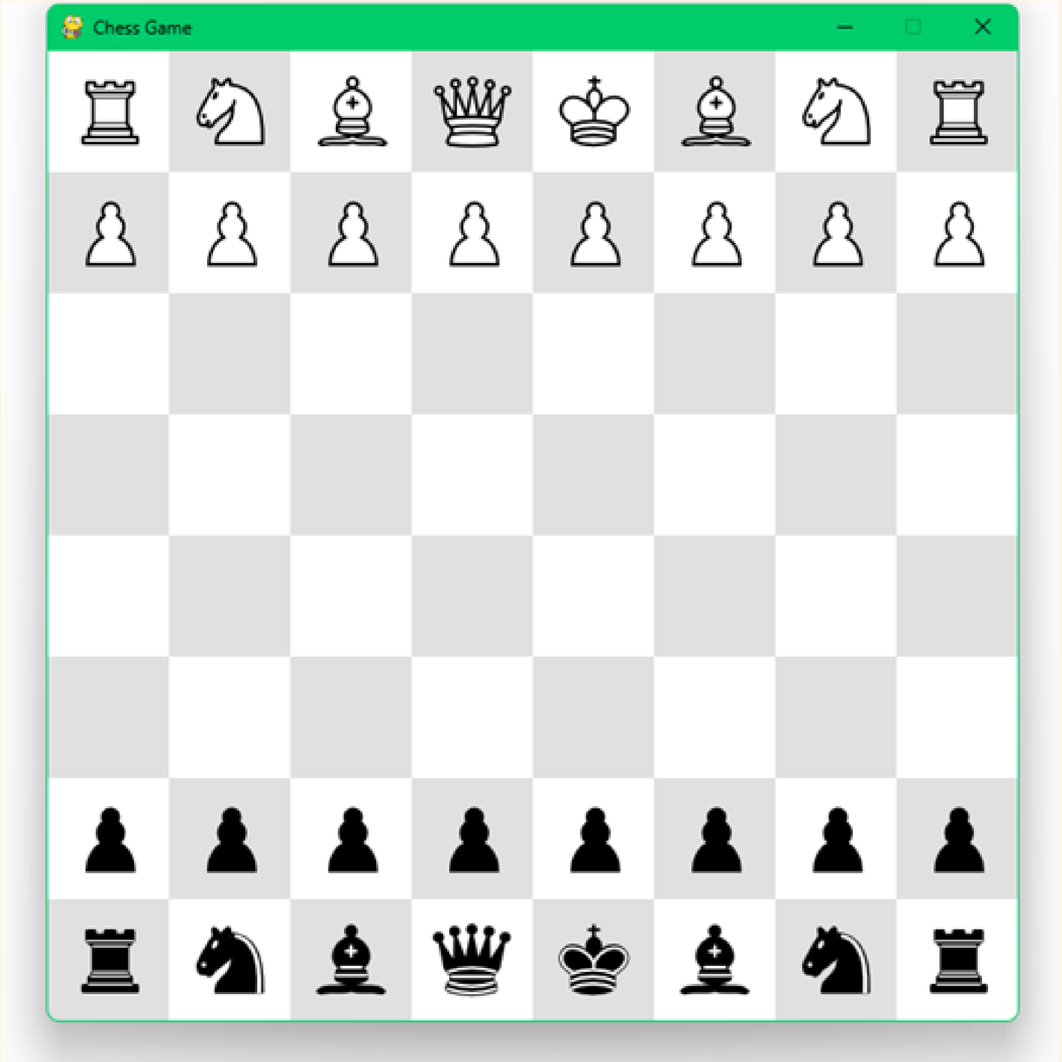




- To **improve the performance** of the AI, we can use techniques such as move orders, which involves evaluating the most promising moves first, and transposition tables, which **store previously evaluated board states** to avoid redundant evaluations.

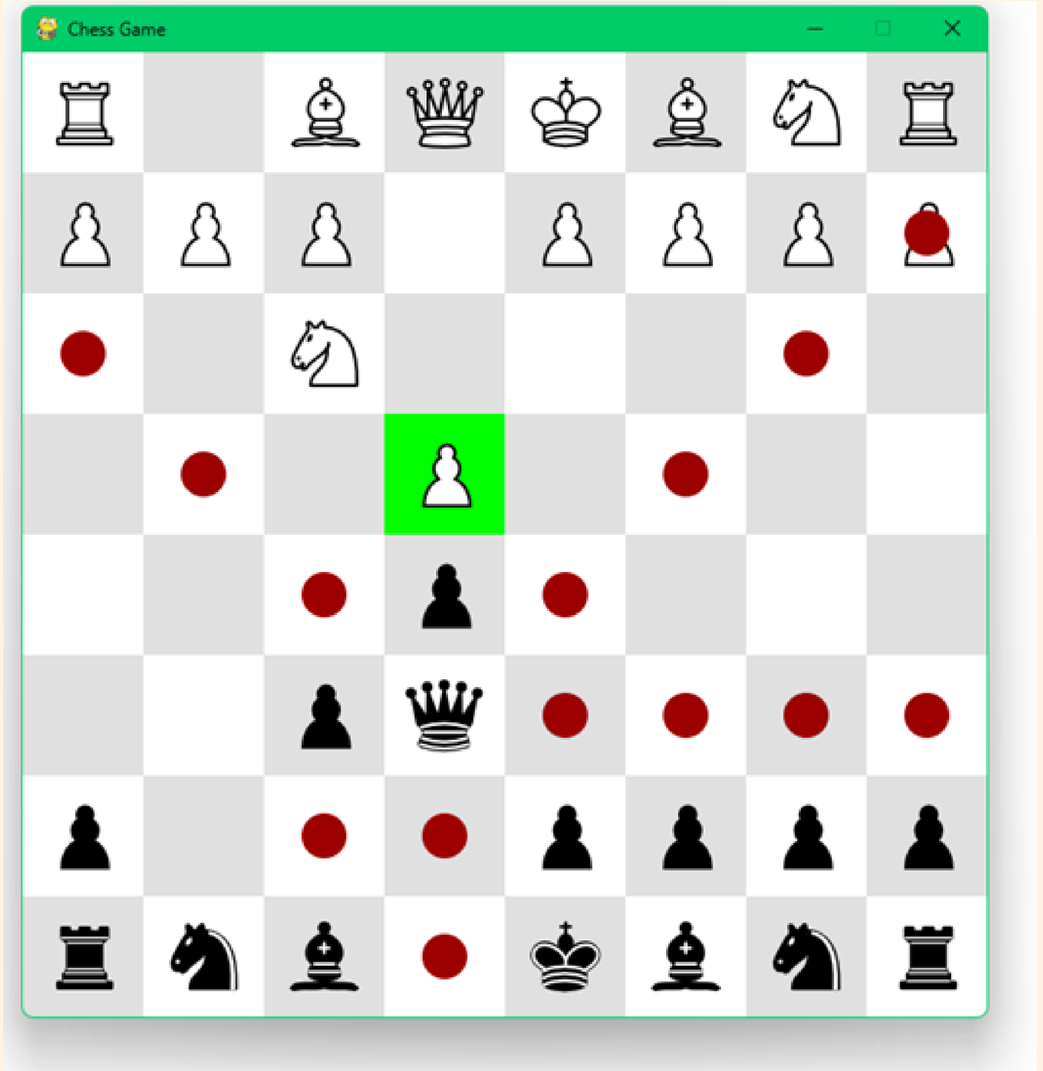
UI

CHESS GAME



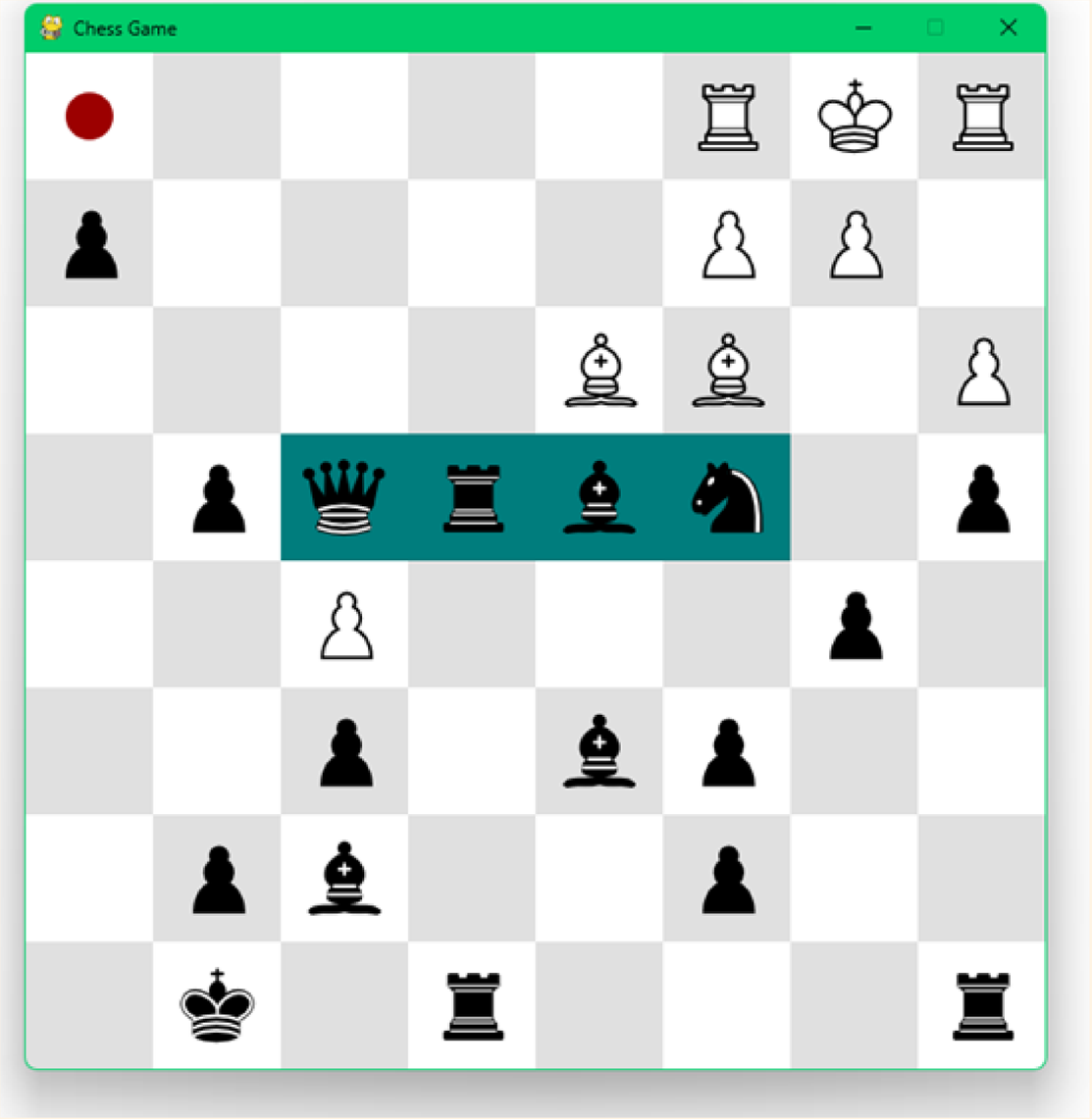
UI

CHESS GAME



UI

CHESS GAME



COMPARE PERFORMANCE

```
Caculating.........
Total nodes 226225
Total nodes found in cache 0
Time: 53.927438735961914
Max: -975.0
0 1 2 2
+ 0 1 2 3 4 5 6 7 +
0 r   b q k b n r 0
1 p p p p p p p p 1
2     n           2
3                 3
4   P             4
5                 5
6 P   P P P P P P 6
7 R N B Q K B N R 7
+ 0 1 2 3 4 5 6 7 +
```

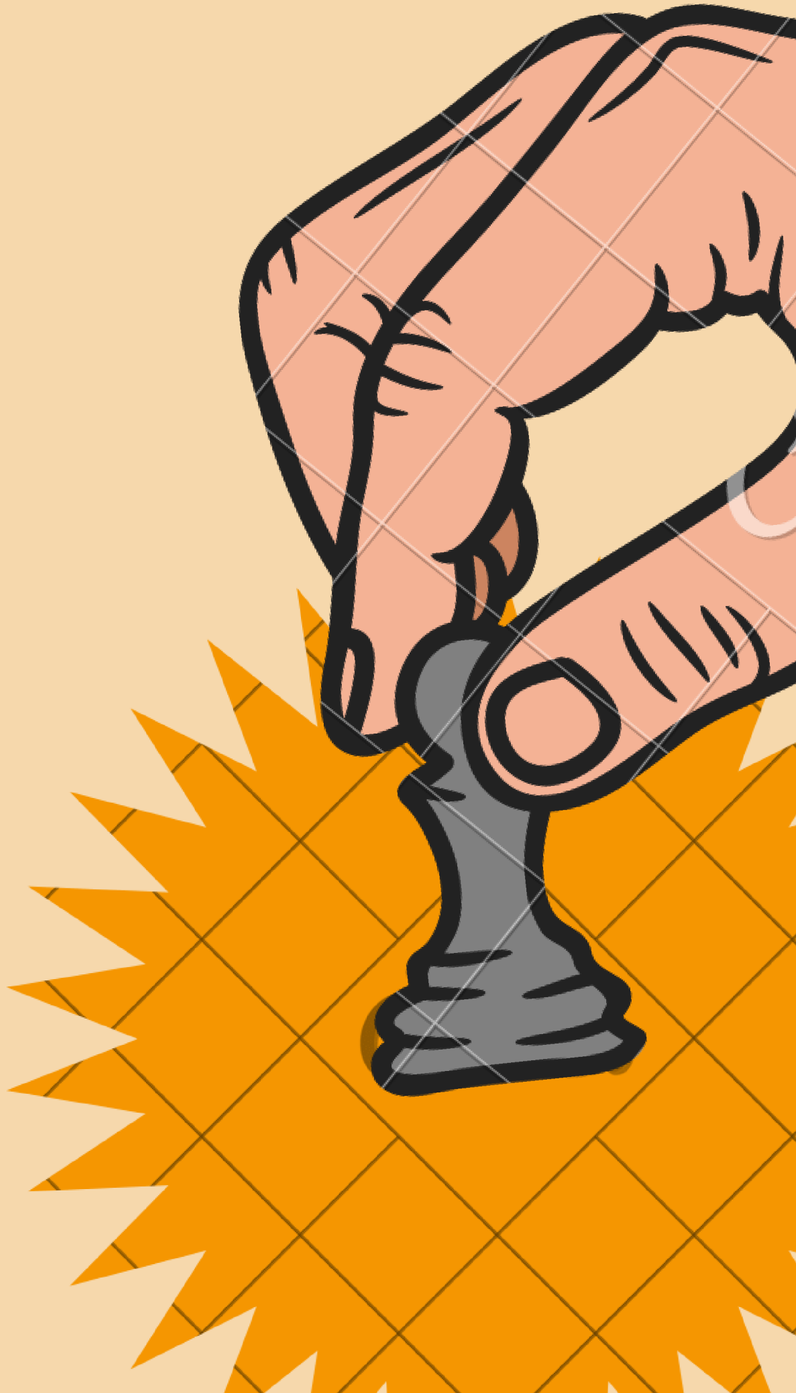
- **Without Alpha-Beta pruning**

```
Caculating.........
Total nodes 31042
Total nodes found in cache 0
Time: 8.05811095237732
Max: -975.0
0 1 2 2
+ 0 1 2 3 4 5 6 7 +
0 r   b q k b n r 0
1 p p p p p p p p 1
2     n           2
3                 3
4   P             4
5                 5
6 P   P P P P P P 6
7 R N B Q K B N R 7
+ 0 1 2 3 4 5 6 7 +
```

- **With Alpha-Beta pruning**

CONCLUSION

- In conclusion, the minimax algorithm with alpha-beta pruning is an effective approach to creating a chess-playing AI.
- By evaluating all possible moves and recursively analyzing board states, AI can make informed decisions about the best move to make.
- Alpha-beta pruning reduces the number of nodes that need to be evaluated, improving the performance of the algorithm.
- With additional techniques such as move ordering and transposition tables, a chess-playing AI using the minimax algorithm and alpha-beta pruning can become a formidable opponent for human players.



THANK YOU FOR LISTENING

LET'S PLAY CHESS!