PYTHON PROJECT (2022-23)
EE-327

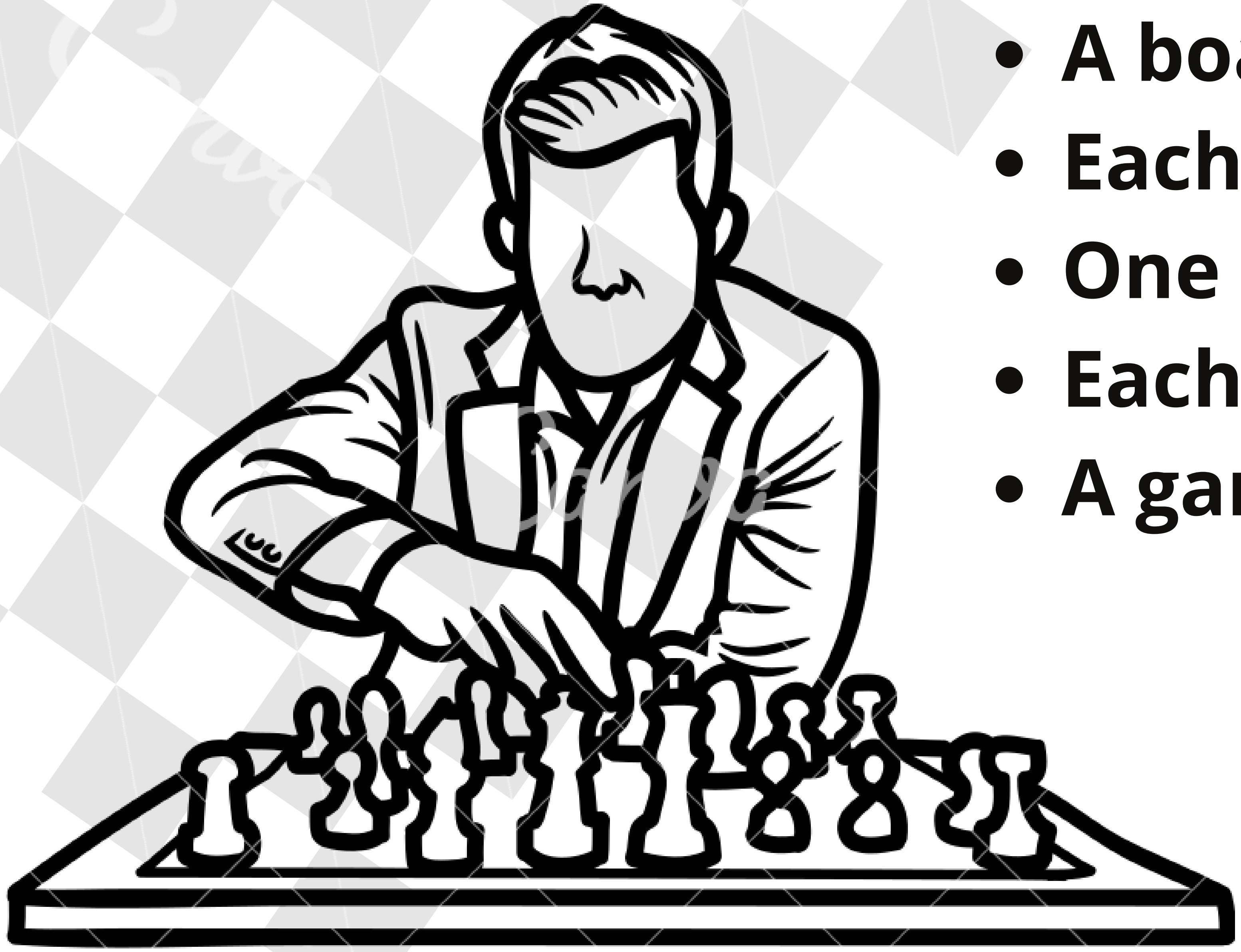
CHESS GAME

GAUTAM KUMAR 2K20/EE/105
HIMANSHU 2K20/EE/122
DISHU 2K20/EE/096

CONTENT

- Chess
- Scope of our project
- Building blocks of chess
- Tools and Techniques used
- Playing steps
- Implementation of program
- Structure of program
- Files
- Further scope

CHESS??

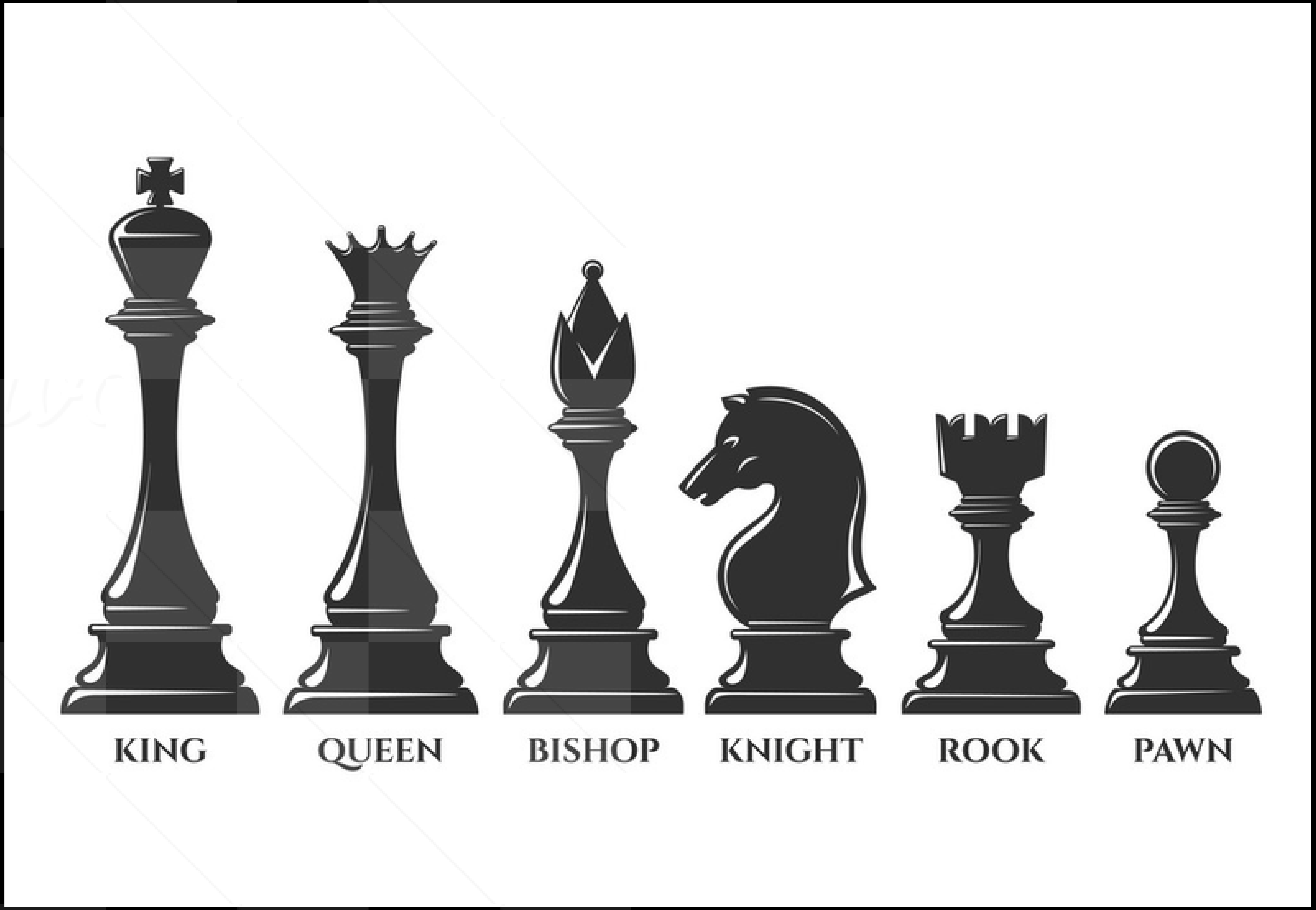


- A board game played by two people
- Each player control an army of 16 pieces
- One army is black, the other white
- Each piece moves in a unique way
- A game of strategy and concentration

OBJECTIVE

The goal of the game is to checkmate the opponent's king by placing it under an avoidable threat of capture.

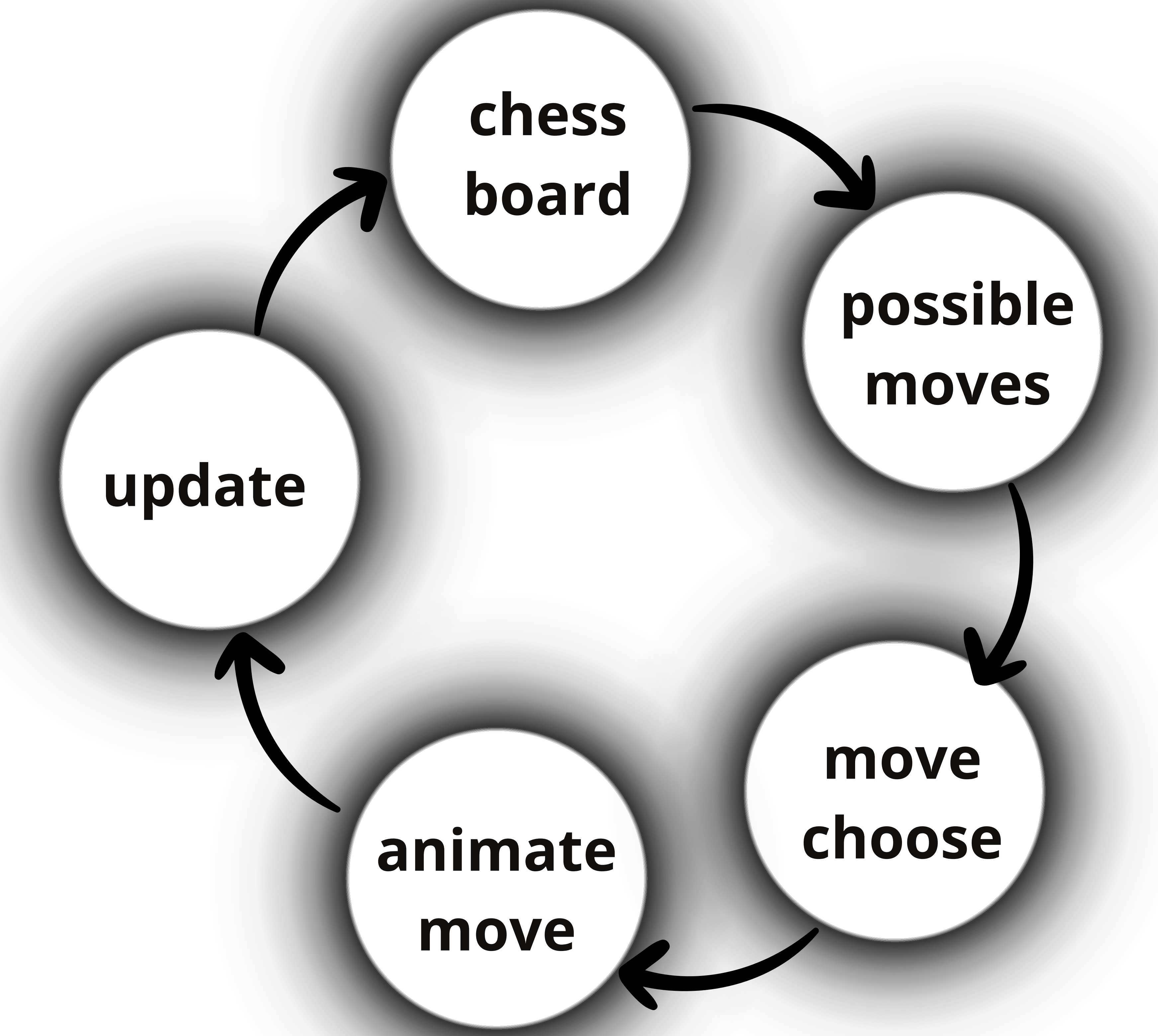
CHESS PIECES



SCOPE OF OUR PROJECT

The scope of our project is limited to two players to play chess as real in computer.

BUILDING BLOCKS OF CHESS



chess board

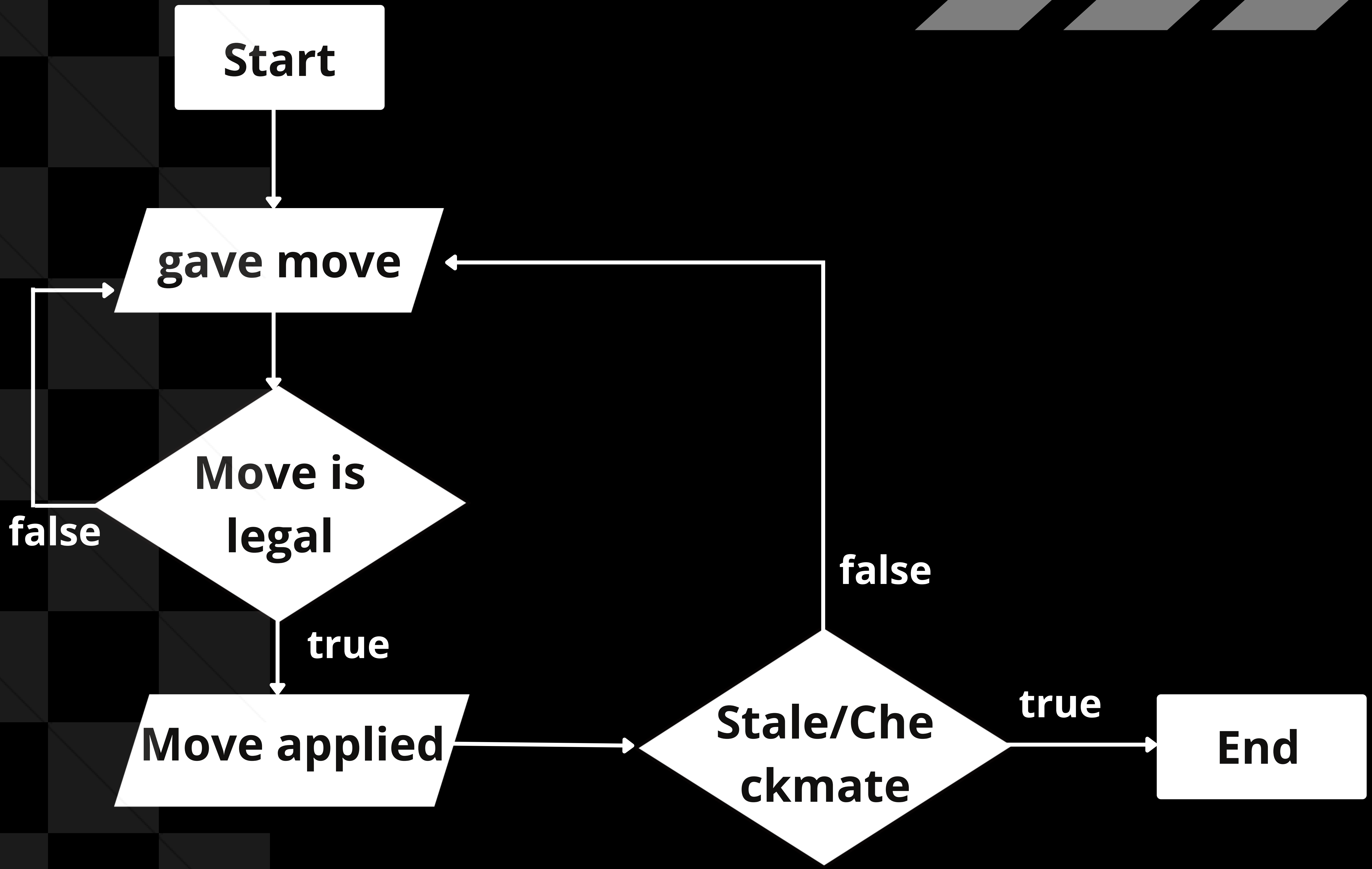
possible moves

move choose

animate move

update

PLAYING STEPS



Start

gave move

Move is legal

false

true

Move applied

Stale/Checkmate

false

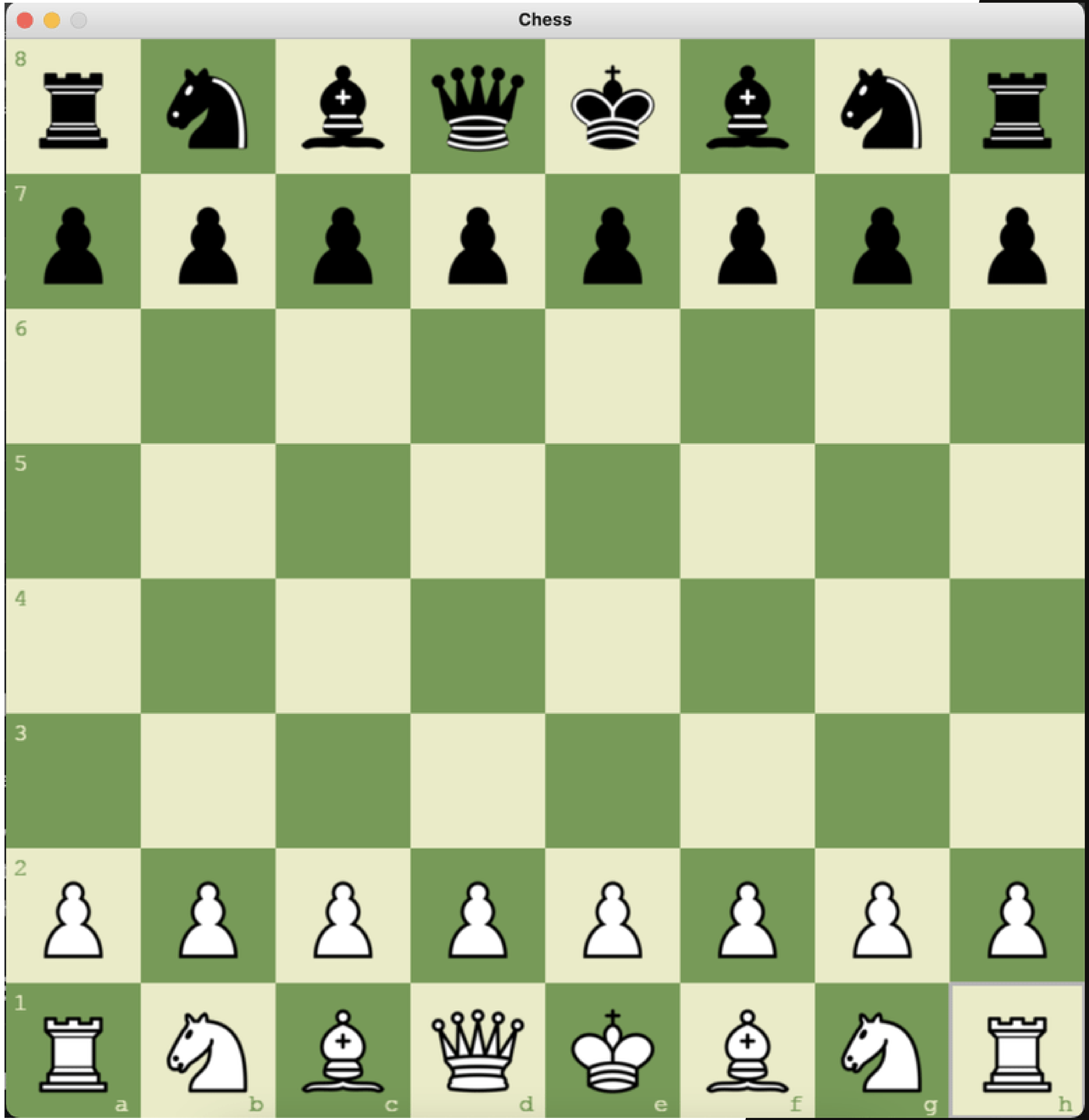
true

End

TOOLS AND TECHNIQUE USED

Python 3 : It is a General-purpose dynamic programming language, which provides the high-level readability and it is interpreted. In our project we use python to calculate the player's move.

Pygame : Pygame is a Python framework for game programming. we use pygame in our project for creating, updating and handling GUI.


Chess
8
7
6
5
4
3
2
1
a
b
c
d
e
f
g
h

IMPLEMENTATION OF PROGRAM

FILES

- **board.py -** used to create and design grid/ board of the game.
- **color.py -** responsible for colors
- **config.py -** responsible for the configuration(font, theme etc)
- **const.py -** save main contants that are needed for the game
- **dragger.py -** responsible for letting us drag any of these pieces all around the grid.
- **game.py -** responsible for all rendering methods
- **main.py -** have attributes - screen and method - main loop
- **move.py -** responsible for saving a move
- **piece.py -** to create the pieces
- **square.py -** part of grid that contains pieces
- **theme.py -** responsible for colors of trace, background, moves
- **sound.py -** it has two references move sound and capture sound

FURTHER SCOPE

we can make it player vs computer (intelligent chess engine)

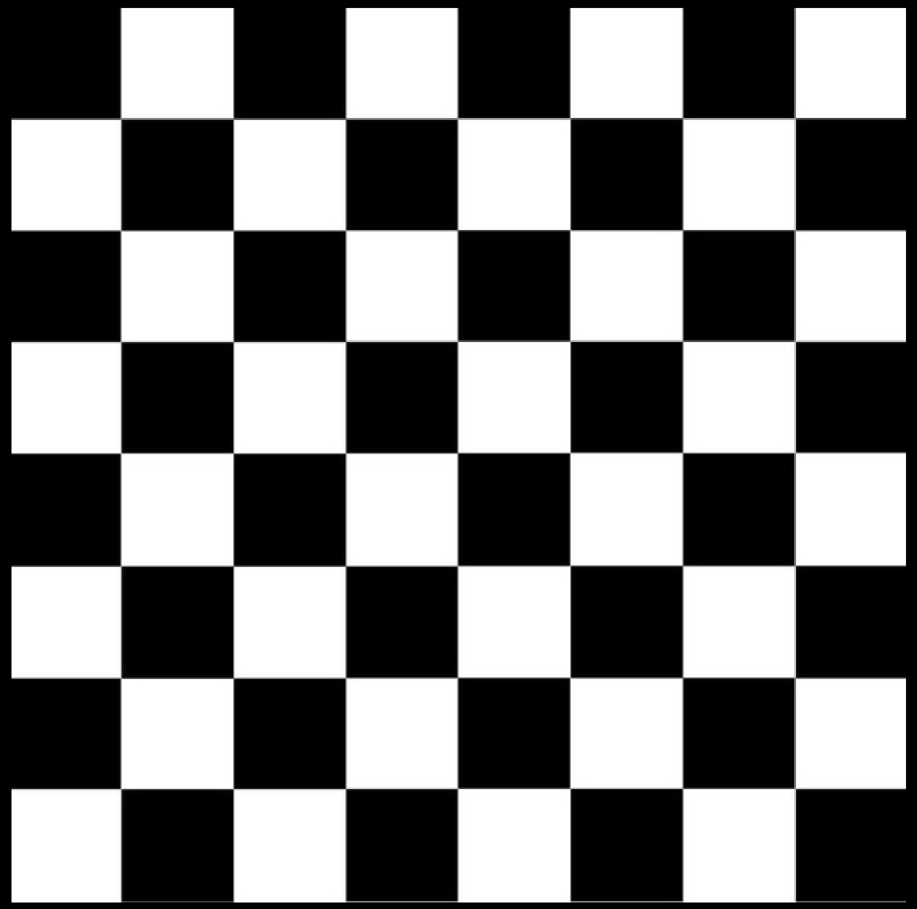
we can introduce two things to computer for making the game intelligent which will make the game to do optimal moves

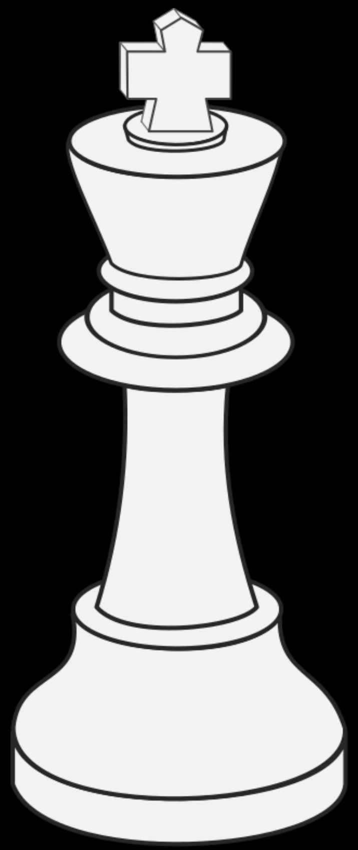
- A technique to choose the move to make amongst all legal possibilities, so that it can choose a move instead of being forced to pick one at random.
- A way to compare moves and positions, so that it make intelligent choices.

Here we need to use some Deep Learning and AI tools like TensorFlow.

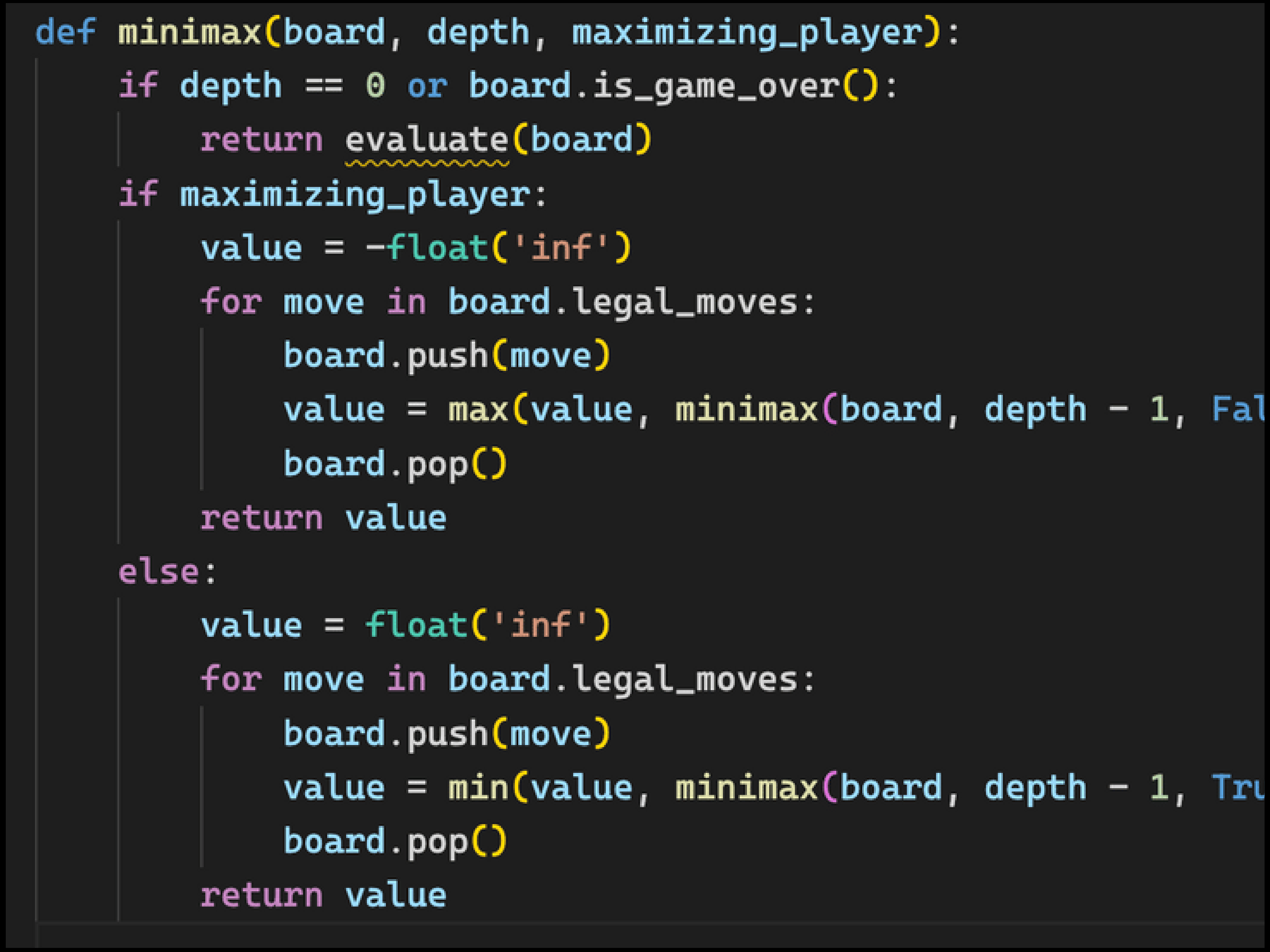
One famous algorithm Minimax can be used to estimate and evaluate the move

Techniques like aplha prunning search and can also be explored and tried to implement in the project.





MINIMAX ALGO

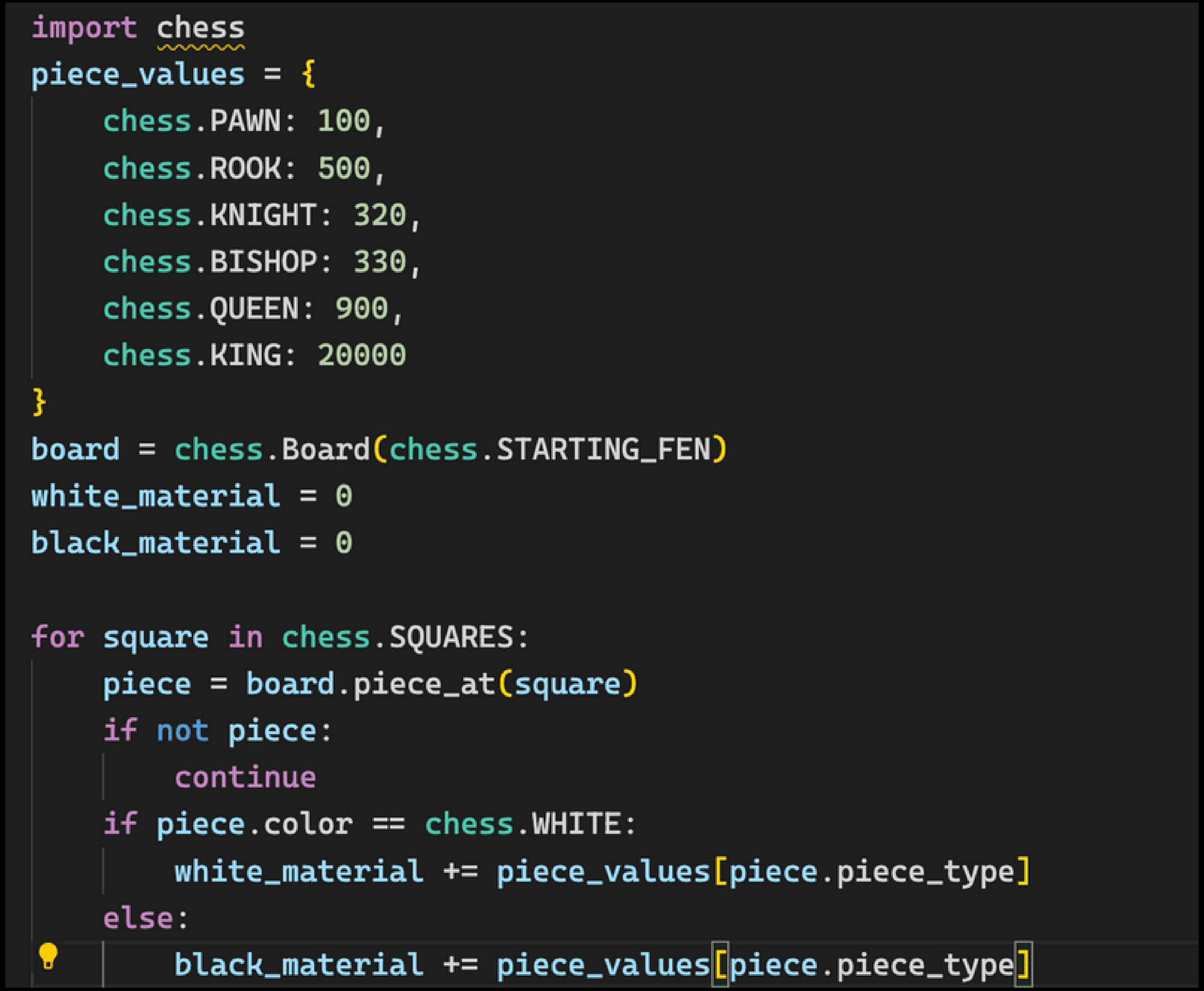


```
def minimax(board, depth, maximizing_player):
    if depth == 0 or board.is_game_over():
        return evaluate(board)
    if maximizing_player:
        value = -float('inf')
        for move in board.legal_moves:
            board.push(move)
            value = max(value, minimax(board, depth - 1, Fal
            board.pop()
        return value
    else:
        value = float('inf')
        for move in board.legal_moves:
            board.push(move)
            value = min(value, minimax(board, depth - 1, Tru
            board.pop()
        return value
```

Minimax is a kind of backtracking algorithm that is used in decision making and game theory to find the optimal move for a player, assuming that your opponent also plays optimally. It is widely used in two player turn-based games such as Tic-Tac-Toe, Backgammon, Mancala, Chess, etc.

In Minimax the two players are called maximizer and minimizer. The maximizer tries to get the highest score possible while the minimizer tries to do the opposite and get the lowest score possible.

Evaluation Technique



```python
import chess
piece_values = {
    chess.PAWN: 100,
    chess.ROOK: 500,
    chess.KNIGHT: 320,
    chess.BISHOP: 330,
    chess.QUEEN: 900,
    chess.KING: 20000
}
board = chess.Board(chess.STARTING_FEN)
white_material = 0
black_material = 0

for square in chess.SQUARES:
    piece = board.piece_at(square)
    if not piece:
        continue
    if piece.color == chess.WHITE:
        white_material += piece_values[piece.piece_type]
    else:
        black_material += piece_values[piece.piece_type]
```

It can be as simple as an indicator for whether a given side has been checkmated, but for limited-depth search this isn’t useful. A slightly more mature estimate is in counting the pieces on each side in a weighed way. If white has no queen but black does, then the position is unbalanced and white is in trouble. If white is three pawns down but also has an extra bishop, the position is likely to be balanced. This evaluation function scores pieces in terms of how many pawns they’re equivalent to.

THANK YOU