SVILUPPO DI UN'INTELLIGENZA ARTIFICIALE PER IL GIOCO DEGLI SCACCHI

Christian D'Alleva

Giuseppe Di Menna

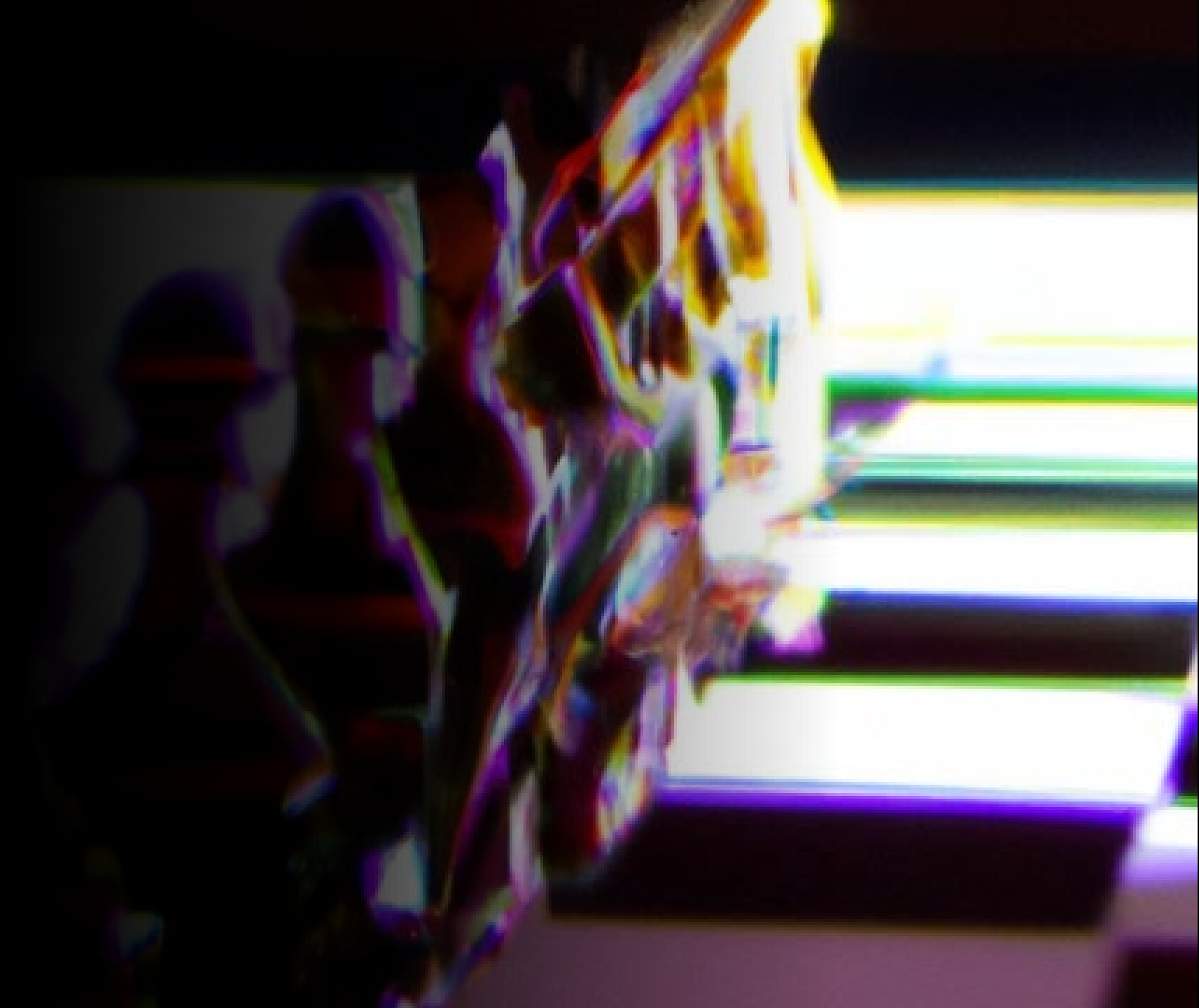
Marco Omicini

"ARTIFICIAL INTELLIGENCE MOVE A KNIGHT PLAYING CHESS, PHOTO" DALL-E by OPENAI

Gli scacchi

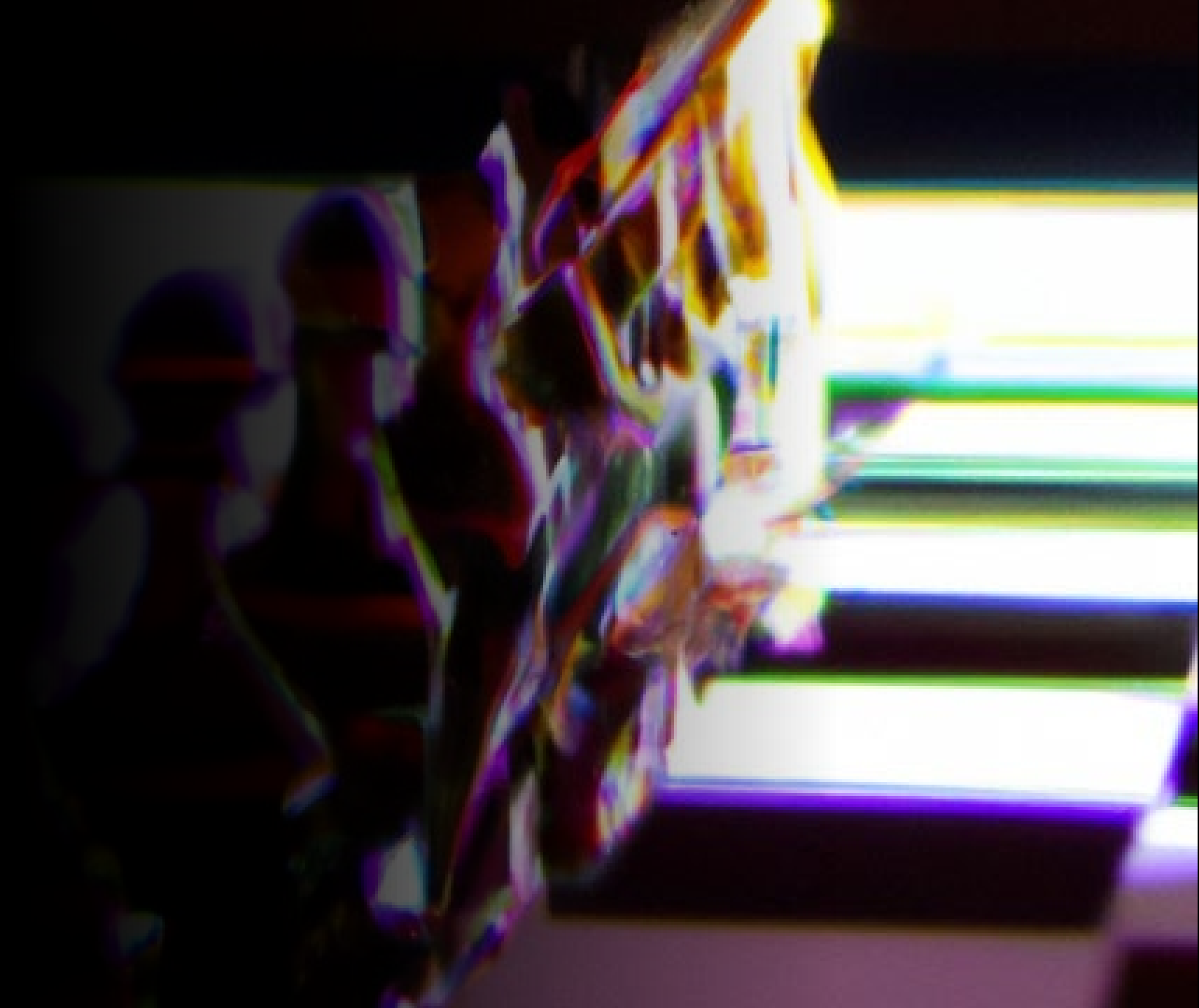
Gli scacchi sono un gioco di strategia nato intorno al VI secolo d.C. e ampiamente conosciuto.

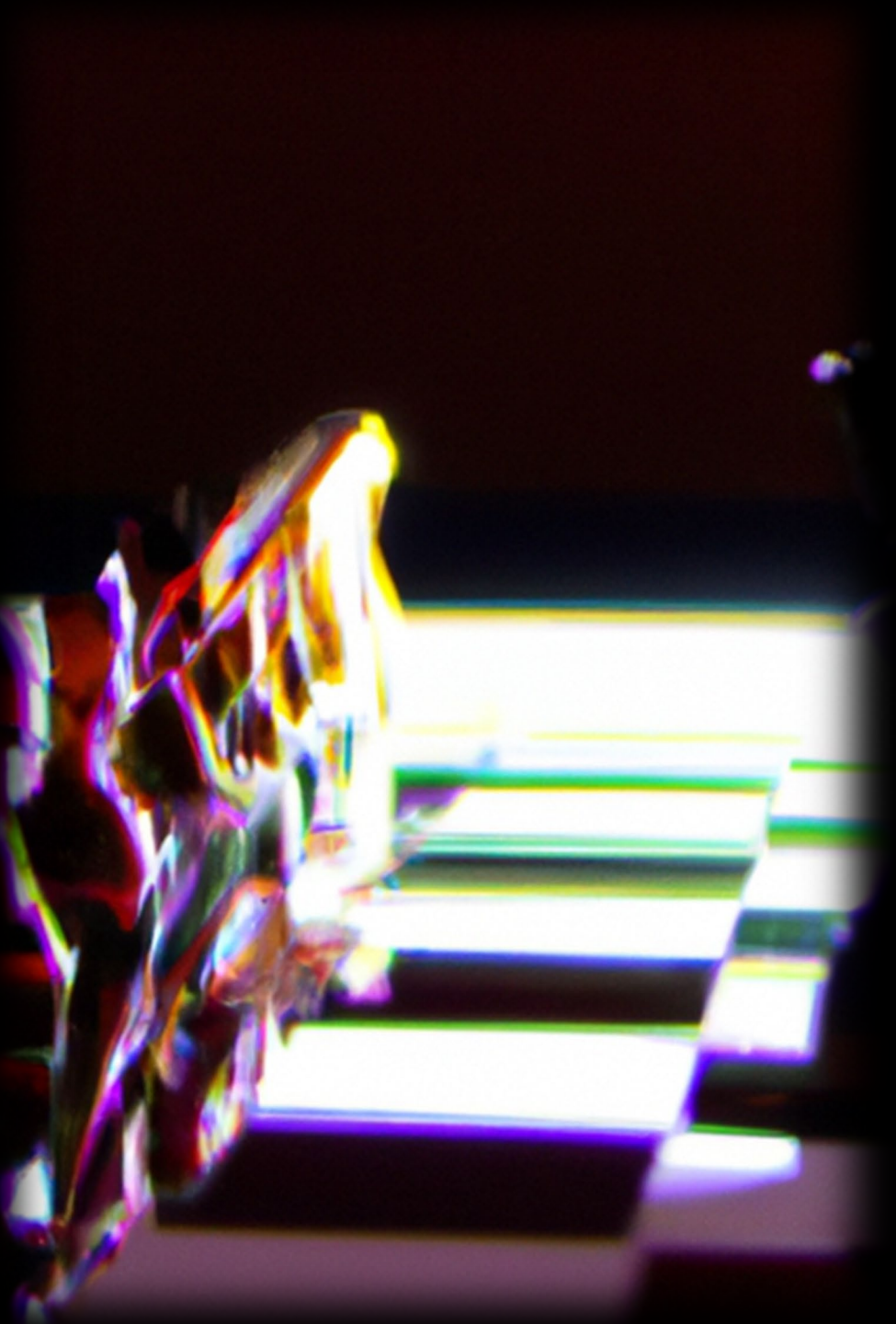
Si stima che le configurazioni possibili sulla scacchiera in partita siano circa 10^{50} : questa vastità di combinazioni rende difficile e interessante creare algoritmi di intelligenza artificiale che si dimostrino di reale efficacia.



Creazione di un motore scacchistico

- Esistono fondamentalmente due approcci per la creazione di un motore scacchistico:
  - MinMax con αβ pruning
  - Apprendimento profondo
- L'efficacia dei due approcci è, ad oggi, pressoché equivalente.
- Il progetto oggetto d'esame si basa sul primo.





MinMax e α-β pruning

- MinMax è un algoritmo di ricerca «depth-first» che si presta particolarmente bene ad essere usato nei giochi a turni con informazione perfetta.
- A causa dell'elevata complessità raggiungibile negli scacchi, MinMax ha l'esigenza di avvalersi di una profondità massima di ricerca.
- La potatura α-β è un'ottimizzazione dell'algoritmo Min-Max che, avvalendosi di due parametri (α e β) e supponendo che l'avversario sarà sempre in grado di trovare la mossa migliore, evita di valutare rami che possono essere considerati a priori non ottimali.

Sviluppo dell’applicativo

Il motore si compone di tre elementi:

- La rappresentazione della posizione
- La procedura di valutazione della posizione
- L’algoritmo di ricerca

SVILUPPO – Rappresentazione della posizione

- La rappresentazione della posizione sulla scacchiera è delegata alla libreria «chess» per Python che, avvalendosi di una rappresentazione binaria delle mosse, permette alle procedure di ricerca e valutazione di essere più snelle e veloci.

SVILUPPO – Valutazione della posizione

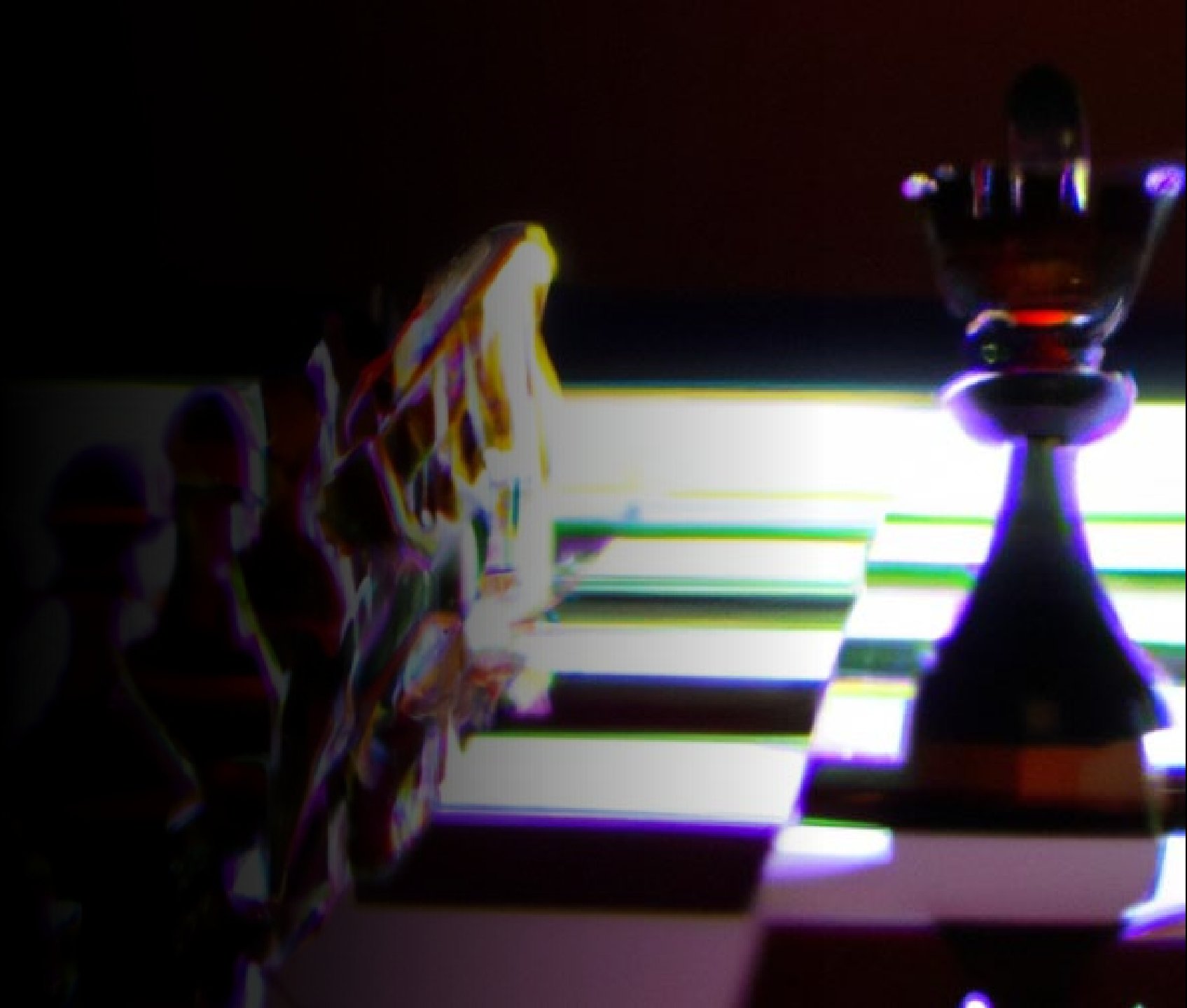
- La procedura di valutazione della posizione tiene conto del materiale presente sulla scacchiera e della posizione dei pezzi.
- Assegna valori positivi ai pezzi bianchi e negativi ai pezzi neri, permettendo così l'uso di un algoritmo NegaMax in fase di ricerca.
- Per la valutazione dei pezzi sono state usate semplicemente delle tavole (come esempio segue quella del cavallo):

-50	-40	-30	-30	-30	-30	-40	-50
-40	-20	0	0	0	0	-20	-40
-30	5	10	15	15	10	5	-30
-30	0	15	20	20	15	0	-30
-30	5	15	20	20	15	5	-30
-30	0	10	15	15	10	0	-30
-40	-20	0	5	5	0	-20	-40
-50	-40	-30	-30	-30	-30	-40	-50

```
def evaluate(self, board:chess.Board):
    if board.is_checkmate():
            return -9999
    if board.is_stalemate():
            return 0
...
 P = len(board.pieces(chess.PAWN, chess.WHITE))
...
 p = len(board.pieces(chess.PAWN, chess.BLACK))
...
mat = (P-p)*100 + (N-n)*300 + (B-b)*350 + (R-r)*500 + (Q-q)*900
...
 if (board.turn):
    return mat+pawns+knights+bishops+rooks+queens+kings
 else:
    return -(mat+pawns+knights+bishops+rooks+queens+kings)
```

SVILUPPO - Algoritmo di ricerca

- Essendo gli scacchi un gioco a somma zero, cioè dove il valore della posizione del giocatore corrisponde necessariamente alla negazione del valore della posizione dell'avversario, si può usare una variante di MinMax che assegna valori positivi alle posizioni del bianco e negativi a quelle del nero, detta Negamax.



SVILUPPO - Algoritmo di ricerca (NegaMax)

```
def negamax(self, board:chess.Board, depth:int):
        bestMove:chess.Move
        if (depth==0 or board.is_checkmate):
            return self.evaluate
        maxScore = -999
        for move in board.legal_moves:
            board.push(move)
            score = -self.negamax
            if(score>maxScore):
                maxScore=score
                bestMove=move;
            board.pop()
        if(depth==self.depth):
            print(bestMove)
            return bestMove
        return maxScore
```

-> se la partita è finita, chiude lo stack delle chiamate

-> aggiunge le mosse del livello successivo

-> chiamata ricorsiva

-> se il punteggio della mossa è migliore, aggiorna la variabile della miglior mossa

-> restituisce il punteggio al livello successivo, o la mossa migliore all'ultimo livello

Questo algoritmo permette di raggiungere 3 o 4 livelli di profondità al massimo, nella slide successiva è mostrata l'integrazione con α β pruning.

SVILUPPO - Algoritmo di ricerca (αβ pruning)

```
def negamax(self, board, depth:int, a:int, b: int):

...

    for move in board.legal_moves:
        board.push(move)
        score = -self.negamax(board, depth-1, -b, -a)
        maxScore = max(score, maxScore)
        if(score > a):
            a = score
            bestMove = move
            if(a > b):
                board.pop()
                break
        board.pop()

...
```

-> α e β si invertono a ogni chiamata ricorsiva (tenendo conto di NegaMax) e hanno valore iniziale

$\alpha = -\infty \ , \ \beta = +\infty$

-> chiamata ricorsiva

-> se il punteggio è maggiore di α sostituisce α

-> se α è maggiore di β, smette di esplorare le mosse

Conclusioni

Il motore così costruito riesce a raggiungere una profondità di 5 o 6 livelli al massimo.

Questa profondità gli permette comunque di giocare a un buon livello.

Anche se non è in grado di battere un grande maestro di scacchi, riesce a raggiungere un punteggio di circa 1800-2000 ELO.

Molto più di quanto basta per essere imbattibile da chi lo ha programmato.