A modified version of Chess

IS 597 – Data Structures and Algorithms

Jiajun Wang, Gibong Hong

Original Chess Game

- 8*8 Table
- Pieces having their own move: Pawns, Rooks, Knights, Bishops, Queen, and King
- Two player game
- Special Rules: En Passant, Promotion, Castling
- Checkmate to win the game

- To make strategic move, we need to consider the potential value of each piece
  ex) King > Queen > Rook > Bishop = Knight > Pawn

  Lots of players follow this logic when they try to capture, exchange, or make other moves.

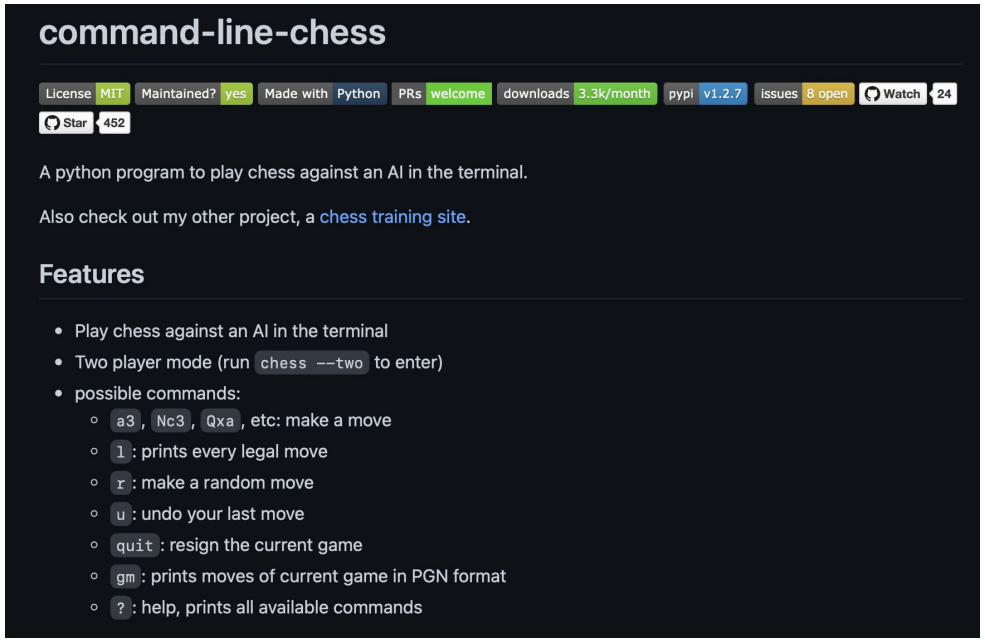
A new version of Chess

Indian Poker Game + Original Chess -> A new version of Chess

- With a deck of cards, every player has one card each, they will stick it on their forehead <u>without seeing their cards.</u>
- Objective: Have the highest card in play to win
- ”Reversed Poker”: As opponents are unaware of their cards, we need to give them impression that they have a high one.

- Each piece is randomly assigned its original value and a certain piece could be capture only if it has lower value than the opponent’s one
- Each player is not aware of values of their own pieces, only to know values of opponent’s pieces.

Baseline Code



command-line-chess

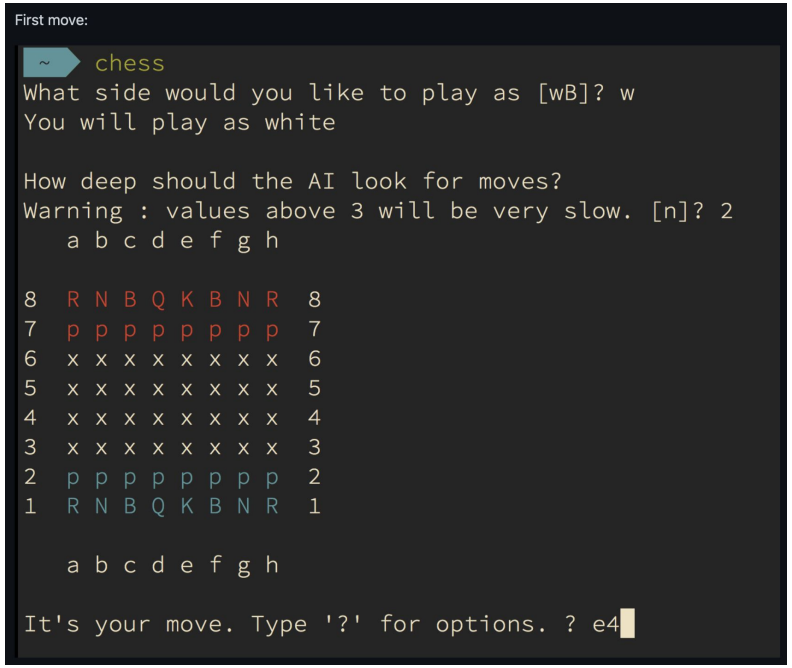
License MIT | Maintained? yes | Made with Python | PRs welcome | downloads 3.3k/month | pypi v1.2.7 | issues 8 open | Watch 24 | Star 452

A python program to play chess against an AI in the terminal.

Also check out my other project, a chess training site.

Features

- Play chess against an AI in the terminal
- Two player mode (run `chess --two` to enter)
- possible commands:
  - `a3`, `Nc3`, `Qxa`, etc: make a move
  - `l`: prints every legal move
  - `r`: make a random move
  - `u`: undo your last move
  - `quit`: resign the current game
  - `gm`: prints moves of current game in PGN format
  - `?`: help, prints all available commands



- Python version of Chess AI
- Each part of the Chess game (Board, Piece, Move, MoveNode, Coordinate) is implemented by Python Class
- We modified original code and added our new idea to make a new version of Chess game

https://github.com/marcusbuffett/command-line-chess

Heuristic Scoring Table

```
class QueensTable:
    def __init__(self) -> None:
        self.table = [
        [-20, -10, -10, -5, -5, -10, -10, -20],
        [-10,   0,   5,  0,  0,   0,   0, -10],
        [-10,   5,   5,  5,  5,   5,   0, -10],
        [  0,   0,   5,  5,  5,   5,   0,  -5],
        [ -5,   0,   5,  5,  5,   5,   0,  -5],
        [-10,   0,   5,  5,  5,   5,   0, -10],
        [-10,   0,   0,  0,  0,   0,   0, -10],
        [-20, -10, -10, -5, -5, -10, -10, -20]
    ]
```

```
# Initialize the Heuristic reward table for each piece
self.tables = {}
self.tables['R'] = RooksTable()
self.tables['▲'] = PawnTable()
self.tables['N'] = KnightsTable()
self.tables['K'] = KingsTable()
self.tables['Q'] = QueensTable
self.tables['B'] = BishopTable()
```

- We need to create the board evaluation part as Minimax algorithm is used for each step.

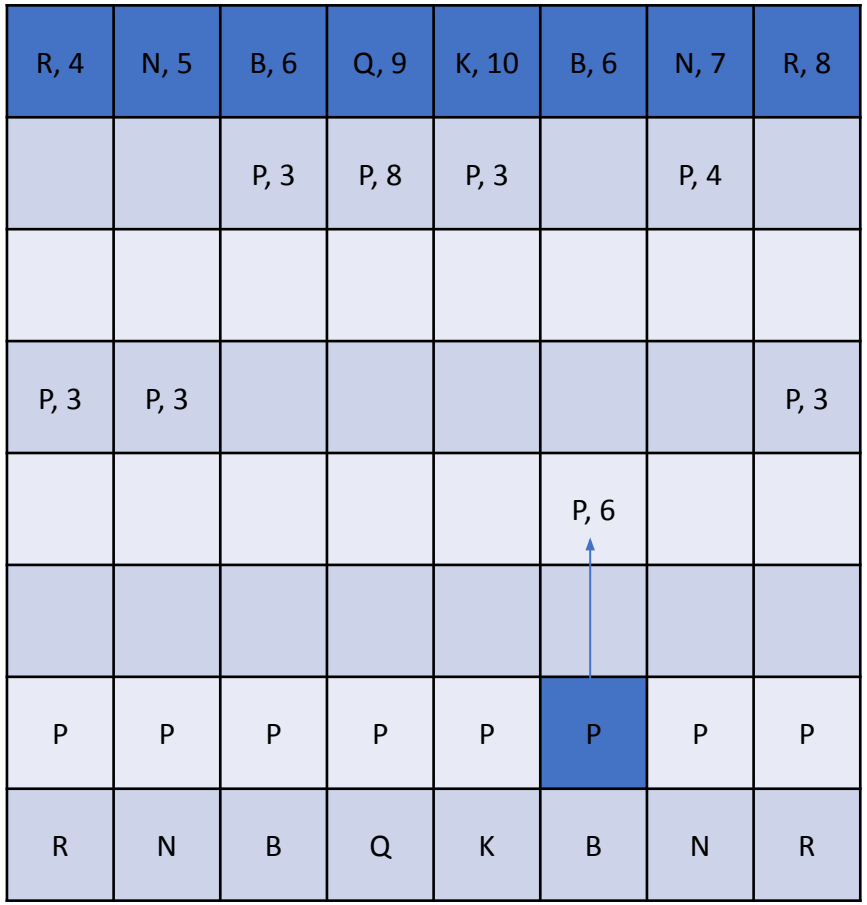
- Piece Square Tables by Heuristic approach

  The values could be set in an 8*8 matrix so that it has a higher value at favorable positions and a lower value at a non-favorable place.

  For example, in the case of Queen piece, she would like her to be placed at the center position
  as she can dominate relatively more positions from the center.

https://medium.com/dscvitpune/lets-create-a-chess-ai-8542a12afef

Incorporate Scoring Table into Evaluation



- Consider the scoring table initialized for each piece.
- Also, we need to include the position of opponents' pieces such that we can adjust the value of scoring table to make reasonable move: Our pieces would rather avoid the opponents' pieces having higher values (6, 7, 8)
- For each turn, scoring function updates the values in the original scoring table by considering possible legal moves of all pieces and finding whether the new coordinate is occupied by the opponent's piece.

- Rule Exception: Queen and King can be captured by any other pieces, to make the game end.

Incorporate Scoring Table into Evaluation

Original static table for Pawn

0	0	0	0	0	0	0	0
50	50	50	50	50	50	50	50
10	10	20	30	30	20	10	10
5	5	10	25	25	10	5	5
0	0	0	20	20	0	0	0
5	-5	-10	0	0	-10	-5	5
5	10	10	-20	-20	10	10	5
0	0	0	0	0	0	0	0

Our table considering the difference of values

0	0	0	0	0	0	0	0
50	50	50	50	50	50	50	50
10	10	20	30	30	20	10	10
5	5	10	25	25	10	5	5
0	0	0	20	20	-5	0	0
5	-5	-10	0	0	-10	-5	5
5	10	10	-20	-20	10	10	5
0	0	0	0	0	0	0	0

Minimax

```
def minimax(self, depth, maximizing):
    if depth == 0:
        return None, self.board.points
    currentSide = self.board.currentSide
    moves = self.board.getAllMovesLegal(currentSide)
    res = None
    if maximizing:
        score = -100
    else: score = 100
    for move in moves:
        self.board.makeMove(move)
        print(move.notation)
        temp_point = self.minimax(depth-1, maximizing == False)
        if maximizing and point > score:
            res = move
            score = point
        elif not maximizing and point < score:
            res = move
            score = point
        self.board.undoLastMove()
    return res, score
```

Demo

Limitation and Further studies

- Need to make scoring matrix learnable by Machine Learning-based training for AI

  AI can decide the values of scoring table for each step by considering the position of pieces.

- Define more strategies based on the remaining pieces of the counterpart.