Heuristic Minimax Chess Player

I will be building a chess game from scratch, and then build a game tree. Then, I will use the tree to build a minimax player for chess. It's likely going to be low-ply, probably around 2 or 3, but I'm going to do as much as I can to optimize the efficiency to make it as good as possible. My plan is to do it mostly in the console, but if I have time I'd like to integrate CMU graphics or a socket of some sort to make the visuals better, or to make different difficulty levels that the player can play against.

Where I am at

The game is able to detect when a player is in check, and I just need to update the legal moves to reflect the in check part. After that, establishing checkmate and pawn promotion then onto the game tree

Similar Projects

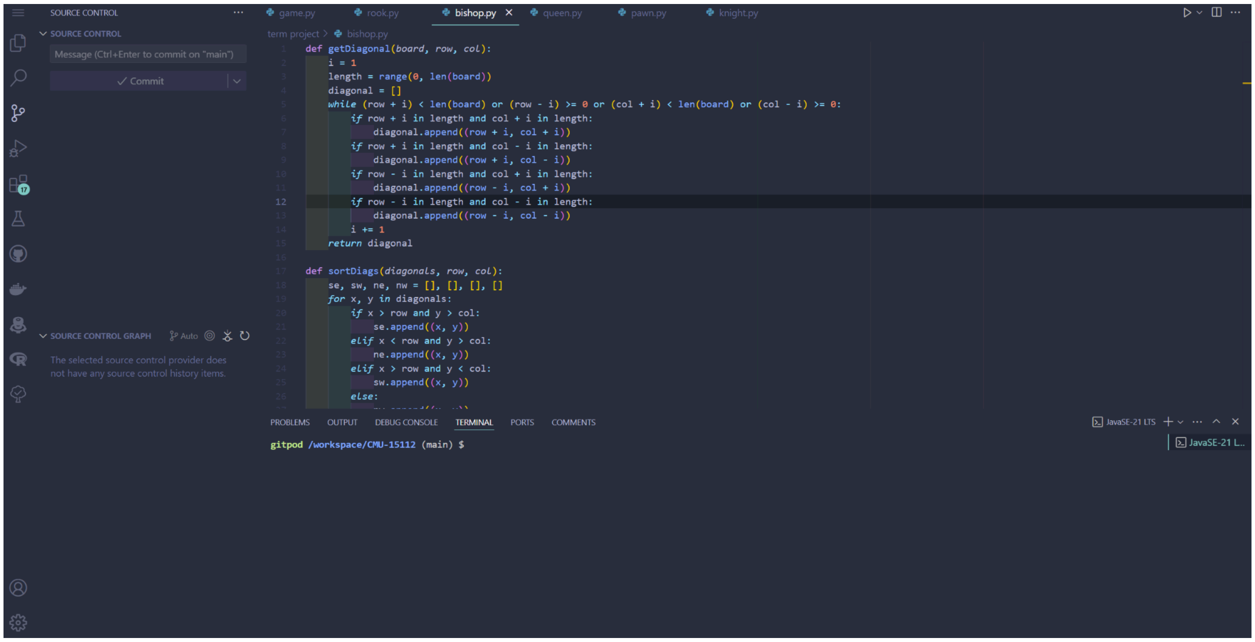
I've done similar projects with tic-tac-toe and Connect 4 in the past, building a game then building a game tree and using it for a minimax player. There are also some similar term projects in the past, such as the Chess Game AI project by Ares Kapoor or Chris Angelov.

Tech List

I'm using pandas to make it look nice in the console, but currently I don't have any external modules that I will be using for my project.

Version Control

I am writing this in a VS Code workspace and committing it to a GitHub repository.



The selected source control provider does not have any source control history items.

```python
def getDiagonal(board, row, col):
    i = 1
    length = range(0, len(board))
    diagonal = []
    while (row + i) < len(board) or (row - i) >= 0 or (col + i) < len(board) or (col - i) >= 0:
        if row + i in length and col + i in length:
            diagonal.append((row + i, col + i))
        if row + i in length and col - i in length:
            diagonal.append((row + i, col - i))
        if row - i in length and col + i in length:
            diagonal.append((row - i, col + i))
        if row - i in length and col - i in length:
            diagonal.append((row - i, col - i))
        i += 1
    return diagonal

def sortDiags(diagonals, row, col):
    se, sw, ne, nw = [], [], [], []
    for x, y in diagonals:
        if x > row and y > col:
            se.append((x, y))
        elif x < row and y > col:
            ne.append((x, y))
        elif x > row and y < col:
            sw.append((x, y))
        else:
```

```
gitpod /workspace/CMU-15112 (main) $
```

Source Code Explanation

Currently I am writing the chess game itself, and I'm working on finding all legal moves on the board. I've finished rooks, bishops, knights, pawns, and the queen, and just need to find moves for the king. The game class is able to make moves and detect whose turn it is, and I just need to implement the rest of the rules of chess (checking, en passant, castling, etc) and I can move on to building the game tree.