Term Project

Abhijeet Prajapati

December 20, 2021

1 Introduction

In this project, I made a chess game and implemented a Min-Max Algorithm with Alpha-Beta Pruning to make moves as an AI. The Min-Max Algorithm makes all the moves that are possible for every piece on the board, but for a game like chess, which has an approximate number of moves of about 10^{123}, there needs to be some sort of extra bit of code that can reduce the number of branches that any given tree spans. That is the Alpha-Beta pruning part.

2 Instructions

2.1 How to compile, Run and Operate

The ChessUI.py file is the "main" file that needs to be run which loads the GUI and loads all the pieces' images from the folder. It will load the ChessGameLogic.py file to handle player movements, checks and etc... The file path will need to be changed to wherever the folder containing the pieces' images are. The board is held in the ChessGameLogic.py file, to test different scenarios, modify that board and the rest of the program will reflect off that. This was coded in Visual Studio Code and sometimes I had a problem occur where the ChessGameLogic.py file was not found since it is not in the same file, in that case I've had to just copy everything in the class ChessGameLogic.py into ChessUI.py and it worked fine.

3 Overview of the System

In my program, I've commented on pretty much every single line explaining what they do and for some I've even provided an example to further illustrate what is happening here, which leaves me with not much to discuss. In that case I'll only further elaborate on how my program has changed from where it first was.

When I first started the project, I had the images loaded in from a separate folder and kept the path in a dictionary, I had a problem with windows file names. Windows doesn't let files be named 'P' and 'p' as it considers them the same. So I changed the file names to also include the color the piece is: 'wP' and 'bP' this later on caused problems when I used char indexing on spaces which are represented with '-' and being only 1 letter long, to quickly solve that issue, I have every empty spot be represent by '–', which is 2 chars long like every other piece is and no longer causes an index error when looking at the 2nd character. The next improvement I set out to make was in how my pieces were being moved.

At the start, I had most of the moves be generated with 1 or 2 for loops filled with if statements making sure the movement is valid for a given piece. To fix that mess, I separated all the piece movement generators to their own methods and then used if statements to check for squares being empty or containing a piece that belongs to the enemy. While I did improve that bit to be more performant and organized, it still have room to go. As I worked on the project, replacing the methods to be better, I realised that I could look in "directions" and use a for loop to modify how far each look up would go and an if statement to keep it within the array bounds. 1

The only piece I figured out how to do that for was the king, some ideas I had about trying was to have a seperate variable to add a variable that kept track of the enemy piece color and check for them

and to modify the i in the for loop to easily change the direction the loop was going in, so that just the 1 loop would be able to find valid squares for both players.

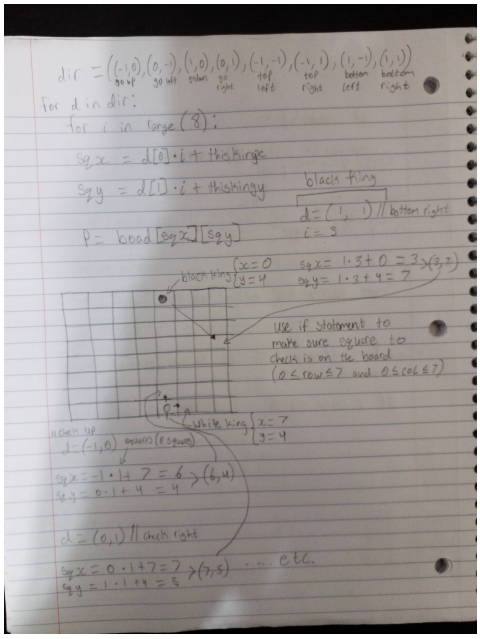

Figure 1: I thought this would look bigger.

Next was the Min-Max algorithm implementation. This was not as complicated as I thought it would be as I've already made an implementation of this in the first assignment. The only tricky part would be how the movement per piece is done and undone. For movement, I had to use 2 for loops, 1 nested within the other, to iterate for each piece: try all of its possible moves then just overwrite the current board with the copy of the board that was made before the move was made to undo the move. Then the Alpha-Beta was also a fairly simple addition, the only part of this that was complicated was how the scoring was handled, I kept getting confused if one score is being checked incorrectly and which color should be minimized and maximized but I think it's done correctly as it has made some fine moves when I've played against it. 2

```
function minimax(node, depth, alpha, beta, maximizingPlayer) is
if depth ==0 or node is a terminal node then
return static evaluation of node

if MaximizingPlayer then      // for Maximizer Player
  maxEva= -infinity
  for each child of node do
  eva= minimax(child, depth-1, alpha, beta, False)
 maxEva= max(maxEva, eva)
 alpha= max(alpha, maxEva)
  if beta<=alpha
 break
 return maxEva

else                    // for Minimizer player
  minEva= +infinity
  for each child of node do
  eva= minimax(child, depth-1, alpha, beta, true)
  minEva= min(minEva, eva)
  beta= min(beta, eva)
   if beta<=alpha
 break
 return minEva
```

Figure 2: Pseudo code for a Min-Max Algorithm with Alpha-Beta Pruning

4 Design Choices and Heuristics

Selecting a heuristic for the chess was simple: if the depth has reached 0, calculate the current state of the board (how many enemy piece are on the board and how many ally piece are on the board) and return it, or check if the previous move (the one that lead to this branch in the game tree) has resulted in a checkmate; and the goal was to minimize the enemy players points and to maximize the AI's and to prevent any checkmates. The biggest improvement that was made to the AI was adding Alpha-Beta pruning to the Min-Max algorithm which greatly reduced the need to travel down a tree that didn't result in a desirable score. Further improvements can be made, such as Threading parts of the algorithm to look at different halves of the possible moves, having it also look for pieces from different ends of the board, and more but I'm not smart enough to be able to do that stuff.

5 Sources

.

1. COSC 3P71 Powerpoint slides by Professor Beatrice Ombuki-Berman
2. Creating an AI that Plays Chess (Video explaning the PseudoCode) Link
3. Alpha Beta Pruning Link