Project Othello

ReadMe

Team members:

Xuefeng Peng	course#:114	Major: CS & Applid Math
Yuntao Zhou	course#:113	Major: CS & Math
Jacob Niebloom	course#:66	Major: CS & Business

Abstract:

In this project, we have built an AI System to play Othello with others. The AI takes a current board layout as input and predicts the next certain number of steps ahead to build up a tree structure for the later usage;and finally, the AI returns the best move we should take as our next move by using Alpha-Beta pruning algorithm. Noteworthily, the Alpha-Beta pruning process consists of a heuristic function, which decides the score for each node in the tree.

Files:

1. Board.java
2. Node.java
3. main.java
4. play

Board.java

Board is the infrastructure of our system. The board records the current board layout by using int[][], the turn, and the move the player takes to reach the current board layout.

The move function takes the responsibility of checking canFlip, making doFile as well as update. Moreover, the getLegalMoves method returns an array list of moves (x,y) that we can take.

Node.java

Node is the fundamental element of the tree structure. Each node contains a board, a layer number, a score, a parent and a child list. The primary functions inside this file is adding child and getting score. The get score function is actually where we put our heuristic function.

Main.java

This class is where we handle the standard input and output related problem. Besides, the depth limit and time limit control are effectuated in this class. The playGame acts as an encapsulation function who will detect which player is the first player and behave accordingly.

The getDecision function is where the alpha-beta pruning process search in the tree that we built with required depth. In the end, this function returns a move (X,Y) for system print out.

The build function is for building up the tree; and this tree will ultimately passed into alpha-beta pruning function.

There also is a function for determining the tree's depth limit by the given time limit as well as the root's child list size. If the size of the child list is too large and the cut-off depth is too deep, then the algorithm will be extremely time-consuming. Beside, out of memory heap error also may happen.

More about the AI, heuristic function and Alpha-Beta pruning:

For pruning:

The alpha-beta pruning is like the common one that will update the alpha and beta in the nodes of trees and the root.alpha's value will be replaced by the possible optimal score.

One thing is different is that I creatively set a if statement at the root to check if there's a move that can take the conner(first-policy) or the edge. Then I will set a special value to both the root.alpha and that special child's alpha for later recognition.
This may make sure that AI won't miss the chance to take the conner.

For Heuristic functions:
We shall consider about 3 aspects to decide our final score of each board state.
1. the advantage on the basic difference of taken positions on the chess board.
2. the possible move for our player on the current board.
3. the number of bad positions that we're less willing to take in early-mid game or the nice position's we're willing to take during the entire game.

For one just the add 100*(maxscore - minscore)/(maxscore + minscore) to total heuristic score.

For the second I counted the legal moves of current board and add it to the H-score without multiplying it. It seems less important than other elements.

For the last one, I put very high positive weight on the conners and positive weight on the edge moves. I put very high negative weight on the diagonal positions(x4) that is next to the conner and other positions(x8) on the edges while next to conners. Also I will check the positions for the other player, I want he to take these positions. His bad positions will count as part of nicescore of my AI. So all of these considerations will be given by the "nice score" and "bad score" which will also be added to the H-score.

More about the Time, Depth and Child list size:

Due to the reason that Java is not a memory flexible language, the deeper the tree the more the nodes we will have;thus, if we build this tree into certain substantial depth, the memory issue appears. So we have built up a bi-directional cut-off system to prevent the memory problem to happen. The first cut-off mechanism monitors the number of the children a node has, if this node's child list size exceeds 10, then the system will automatically and randomly cut it to 8 since the child number growth rate is exponential. Furthermore. the second cut-off system controls the depth by a given time limit. The specific time, depth and child number relations are listed below:

time	depthLimit
1s	3
4s	if childList num<7, depth=5, else depth=4
16s	if childList num<6, depth=6, if childList num=7or6 depth=5, if childList num=8 depth=4
60s	if childList num<6, depth=6, if childList num=7or6 depth=5, if childList num=8 depth=4
240s	if childList num<6, depth=6, if childList num=7or6 depth=5, if childList num=8 depth=4

However, even though the bi-directional cut-off system makes the AI's action more flexible and so as to enhance its smartness, the memory constrain is still the fetter for us to build the tree deeper and see more forward. As for the time limit 240s, the AI actually definitely would not spend that much time to return our move.

Credits List:

Yuntao Zhou:
Alpha-Beta Pruning
Heuristic Score Function
Debug

Xuefeng Peng:
Tree structure
Node structure
Platform Connection System
Time, depth limit system and pass functionality
Debug

Jacob Niebloom:
Board system
Play file
Debug