Chunhao Li (Frederick)

Monday 10:00 - 12:00

Tutor: Kam Wong

U6527752

Introduction

This assignment requires us to write an AI which can play the game Othello. The main algorithm I use is minimax with an optimisation which is known as Alpha Beta Pruning. The timeout for two players is the same, thus, a strong bot needs to use a good heuristic method and can calculate fast.

Process

The main structure I use is the game tree. The type of the nodes is my own defined type `GameState` which is the tuple `(Score, [Move], Game)`. `Score` is an alias of `Int` while Move is an coordinate `(Int, Int)`. Apart from the bot `“helloWorld”`, the first bot I wrote is `“greedyBot”` which can pick the next move that can give a good board position. The heuristic I use is the pieces’ board positions and mobility. Good board positions are some positions where pieces may or cannot be

flanked by the opponent. Therefore, the corners of the board I give highest weights and the three positions next to a corner I give lowest weights which are negative numbers. The weights of other positions are given in the range of lowest to highest. And the score of a board is the sum of all AI's pieces with corresponding weights minuses the sum of all opponents' pieces with corresponding weights. The mobility is how many moves AI and its opponent can make. More mobility means player has better board positions. I calculate the difference between two players and gives it a weight of total scores. The "greedyBot" is the AI which finds the move that gives best positions but only in one step. After that, I wrote a stronger bot called "minimaxBot" which can look several turns in the future with the minimax algorithm. The heuristic of it is the same as "greedyBot". The difference between these two AIs is that "minimaxBot" uses the tree structure. I defined several functions which work together to make a tree. The main function is othelloTreeGameState. To make a

tree, first, I use the root node to generate the first layer of GameState with the function ValueNextPositions. Second, to make subtrees of the first layer and subtrees of the subtrees constantly, I defined a function inside othelloTreeGameState called nextGameStateList, which takes GameState as input, and outputs the list of GameState of the next layer. With the help of the function gameTree, I can use nextGameStateList to recursively generate the new GameState lists. Also, I used some other functions valuePositions, newMoveList and nextGame which can generate scores, moves and games. With all these functions, I can finally make a Tree GameState. The last step of finding a move is using the minimax algorithm. In this step, I use functions prune, maximise and minimise. These help to find a node from the tree. However, to get a specific move, I define a function snd' that takes the second element from a tuple of three elements, and thus I can get a list of Move. Then, I use last to get the last move which is what I want. This is the general idea of how AI selects a move. Furthermore, to optimise the minimax

algorithm, I take the method Alpha Beta Pruning which can prune some nodes that may not be searched through and wrote a bot called “alphaBetaBot”. The only difference of minimaxBot and alphaBetaBot is the alpha beta pruning. In terms of pruning, I defined another version of maximise and minimise called alphaBetaMaximise and alphaBetaMinise. These functions take a potential minimum or maximum value and use it to prune the branches that maybe unlikely to be searched through after comparing to the potential value. The strongest bot is “default” which uses two heuristic methods: positions and mobility. The algorithm of it is minimax with alpha beta pruning.

Testing and debugging

The most important way of testing is playing the game with different AIs. I wrote several AIs which should be stronger than the previous ones. The first issue I encountered is when “greedyBot” plays against “helloWorld”. The previous heuristics I applied is based on the

current score. Thus “greedyBot” cannot win when it is the player2. However, my assumption is that it should win because “greedyBot” has heuristics. After inspecting all functions, I found the problem is the wrong heuristic method. I searched for some information about Othello and finally changed heuristics based on positions. The second issue is after finishing “minimaxBot”, I let it play against “greedyBot”. The assumption is the same: “minimaxBot” should win because it can look at several turns in the future which should give a more accurate evaluation. However, when the player1 is “minimaxBot”, it always lost the game. After examining the function valuePositions, I found that I omitted the case when the opponent’s pieces on the board. Since Othello is a zero-sum game, the weights of its opponent should be negative of the player’s instead of 0. After writing the simple AI with alpha beta pruning, I wanted to improve it and get a stronger AI. To reach the goal, I added a new heuristic method of evaluating mobility. The main issue is that how to use this method properly, when to use it. At first, I tried to

use it in the first 30 steps. I tested it with the command “cabal run othello -- -p default -P alphaBetaBot -T console”. Runing in the console mode can see the lookahead of each move. Since these two bots both have the same minimax algorithm with alpha beta pruning, the lookahead should be almost the same. However, calculating steps made the search process run slowly. And thus, I chose to use the combination of two methods through the whole game. Another problem is about the weight of mobility. This can partly decide whether an AI is good or not. To reach a relatively accurate evaluation, I tested different weight and chose 15 as my final weight for mobility evaluation.

Improvement

There are two main directions to improve my AI. Firstly, I can optimise the algorithm. For example, sorting the score properly before pruning which can make it accurate. Some other optimisations that can be applied like

narrowing searching windows which make pruning efficient. Secondly, I can improve my heuristics. Apart from positions and mobility, stability is a significant factor as well. Some advanced heuristics which focus on specific pieces positions can also enhance the AI.