COMP424 Artificial Intelligent Final Project

Yue Xiao

260740996

April 11, 2019

Abstract

Pentago-Swap is a two-player game played on a 6×6 board, which falls into the Moku family of games. There are wide range of algorithms such as Monte Carlo tree search, minimax search and minimax search with alpha-beta pruning to help AI agent to find the best moves.As presented in my code, I've applied minimax search with alpha-beta pruning to acquire less run-time and better performance on my AI agent.

1 Introduction

Pentago-Swap game is a two-player game played on a 6×6 board, which is the game that falls into the Moku family of games.The board in this game is divided into four 3×3 quadrants, and two players are represented with white and black. The moves consist of two phase which are placing and swapping.The goal of this game is to try to achieve 5 pieces in a row(horizontally,vertically or diagonally)before their opponent does.

2 Motivation and explanation of Related Work

In my submitted code, minimax search with alpha-beta pruning is the chosen algorithm. Minimax search is a decision rule used in most artificial intelligence area for minimizing the possible loss for a worst case (maximum loss) scenario. Therefore it is perfect for our game since the goal of the two-player game is to optimize the winning score for my AI agent by choosing the best move and try to minimize opponent's next best move. And the reason to try alpha-beta pruning is that the full minimax search explores some parts of the tree it doesn't have to.So alpha-beta pruning is a way of finding the optimal minimax solution while avoiding searching sub-trees of moves which won't be selected since When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that

cannot possibly influence the final decision.

In order to make the code clearer and easier to implement, all algorithm-related codes are included in MyTools.java, and in Student_Player.java, the thing I do is to get the current player_id and then call the minimaxAlphaBeta function in MyTools.java to choose the best move.

3 Brief description of the theoretical basis

As stated in Wikipedia, [1]A minimax algorithm is a recursive algorithm for choosing the next move in an n-player game, usually a two-player game. A value is associated with each position or state of the game. The player then makes the move that maximizes the minimum value of the position resulting from the opponent's possible following moves. The way minimax algorithm do is to expand the complete search tree first, then start from leaf nodes back track assuming both players play optimally and picking the best move. [2]And for alpha-beta pruning,it is a search algorithm that seeks to decrease the number of nodes that are evaluated by the minimax algorithm in its search tree. It is an adversarial search algorithm used commonly for machine playing of two-player games.

4 Proposed Approach

Minimax search algorithm is a recursive algorithm for choosing the next move The general idea is that I built a search a tree with depth 2 where nodes are all kinds of possible legal moves combinations by clone() and processMove() and pass along initial alpha and beta values.Furthermore the current move of AI is the root of the search tree. Among all combinations, evaluate the best move on my AI agent, and the worst move on its opponent.

In Student_player.java, player_id is obtained by calling getTurnPlayer() function to make algorithm work,and then define the maximum depth for the algorithm to search which is 2 in my code to prevent run-time error. And in the search tree for pentago-swap game, there are two kinds of nodes, nodes representing AI agents' moves and nodes representing your opponent's moves. The first thing I do related to the algorithm is to get all legal moves of current state and loop through the moves.And in the loop, clone the board state of current move. Since I know when student_player function is called, the turn is on AI's side, so we need to process to opponent's move and for every move call the minimaxAlphaBeta function to get better result. The reason to call the function for every move is to get all legal and possible result, and evaluate the utility. Then we initialized the best score to be the minimum integer, if there is larger score returned from minimaxAlphaBeta function,the best score and the best move will be updated.

In minimaxAlphaBeta function, the main algorithm is designed. First when getting into this function, I check if the node is leaf or if there is no node left. If one of the above two conditions is true, then we go to the heuristic evaluation function of minimax and the function returns a value that is associated with the state board, which indicates how good it would be for a player to reach that position. Then we pass the value to next player and check if the player is the one we need to maximize or minimize. If it is the maximizer player, update alpha value if possible, otherwise update beta value. Then process to next move, and call minimaxAlphaBeta function to evaluate.

In my heuristic function, firstly I initialize two values to count the utility of two players, countPiece is for my AI agent, and countAnother is for the opponent. The I check four sides including horizontal row, vertical row,left-right diagonal row and right-left diagonal row.I loop through the rows and columns accordingly in order to prevent out of range exception, if there are two more same colors for my AI agent, the utility for AI agent is incremented.

5 Pros/Cons

5.1 Advantages

1. it is simple to implement.
2. minimax search using a heuristic evaluation function is a successful technique in a turn-based, two player computer games.
3. Alpha-beta pruning reduces the time complexity from $O(b^m)$ to $O(b^{m/2})$ if it is perfect ordering.
4. Good child ordering improves effectiveness of pruning
5. Alpha-beta pruning will not affect the final result.
6. minimax is useful because it is a reasonable expectation that the opponent can figure out what its best options are

5.2 Disadvantages

1. Run-time of running minimax is slow, since calculating minimum and maximum and compare every value is time-consuming.
2. If the depth of the search tree I define is large, there will be connection error, since the the larger the depth is, the harder AI to make moves.

3. Sometimes it is impractical and inefficient to explore whole search tree.

4. Minimax search will find the best optimal solution only under the condition in which the opponent is perfect.

5. The main disadvantage of using Minimax algorithm is that we need to define two different heuristic functions.

6 Improvements

The main disadvantage of using Minimax algorithm is that we need to define two different heuristic functions. The connection between these heuristic is that, the better a state of a game is for one player, the worse it is for the other player. So we could update the heuristic function by[3] producing only one single heuristic function but doing same work such as negamax algorithm. Secondly, Since the complexity of minimax relies heavily on the branching factor, Reducing branching factor could be another improvement by updating pruning algorithm. What's more, instead of calling Minimax with hard-coded max depth, iterative deepening could be applied to reduce run-time.

7 Other approach: Greedy algorithm

I've also tried the greedy algorithm. Instead of calculating utility of every states and getting into recursive algorithm, This algorithm will just make current best local decision by choosing the move with maximum utility. The advantage of this algorithm is that it requires less run-time and more efficient to find a solution. But the problem is that this algorithm the answer might not be the most optimal decision and only if locally optimal choices lead to a global optimum and the sub-problems are optimal, then greedy algorithm works[4].

References

[1] Minimax *https://en.wikipedia.org/wiki/Minimax*

[2] Alpha-Beta prunning *https://en.wikipedia.org/wiki/Alphabeta_pruning*

[3] improvement of minimax algorithm
https://www.tutorialspoint.com/artificial_intelligence_with_python/artificial_intelligence_with_python_gaming.htm

[4] advantage and disadvantage of greedy algorithm
http://bluehawk.monmouth.edu/rclayton/web-pages/f08-306/greed.html