Jupiter Whitworth and Jacob Gamez

Prof. Guillen-Castrillo

CS441 – Artificial Intelligence

1 November, 2015

Team Project Report – Connect Four

The aim of our project was to use concepts of artificial intelligence to simulate an opposing player in a game of Connect Four. The Connect Four environment is a “board” with seven columns and six rows. Both players in the Connect Four game have the same perspective on the board, meaning the board has a non-arbitrary “top” and “bottom”. For each ply, a player chooses a column and a “token” that indicates that player is placed in the bottommost available row in that column. For our program, the board is represented by a two-dimensional array and we simulated the tokens as integers representing the player number (1 for player one and 2 for player two). Rows that are available are indicated by “zero”, so all nonzero locations are taken and cannot be used in a turn. The object of the game to have four consecutive tokens of one’s indication in one “win lane”. We define a win lane as any possible straight line that can be drawn on the board that have a possibility of winning (has at least 4 spaces). Thus, win lanes can be constituted by rows, columns, and diagonals rising in either direction, though not all diagonals have four possible token locations so not all diagonals are win lanes. When four consecutive tokens for one player are in one lane, the board is said to be in a “win state”. After each ply, the board is evaluated and, if it is found to be in a win state, the player who moved last is determined to be the winner and the game is over.

In order to simulate an opposing player, we had to give the program a way to decide which column to choose in any situation. This necessitated the implementation of a tree data structure in order to model the possible future decisions made by each player, with nodes containing a possible state of the board, alpha and beta values, the node’s depth, and the “minimax value”, or heuristic value. The top “root” node represents the current board state, then it branches off into all possible decisions the next player could make, which manifests as nodes representing the board state given each possible decision. Since there are seven columns and the decision the player has to make is which column to place his or her token in, that means there are seven possible decisions and, therefore, seven nodes branching off from each parent node—one for each column. The depth, or generations of nodes that we go down, also relates to the difficulty setting of the game, as it determines how many ply ahead the computer will predict and factor into its decision making. The depth is defined globally in our program as “DEPTHLIMIT”, so increasing this number inherently increases the difficulty. We found a depth limit of 8 to be a good limit for moderate difficulty, as that way it predicts 4 full moves ahead—the minimum number of moves in order to win.

In order for the program to actually make a decision on which column to choose, it needs to know the desirability of each choice it has so that it can choose the most desirable one. To this end, we apply the Minimax algorithm, whereby we minimize loss in the worst case scenario by taking into account that the opponent will also be choosing the best choice for him or her. For this algorithm to work, it is necessary that the leaf nodes are evaluated for their desirability and given a value that indicates that relative desirability. That method of evaluation is our heuristic, and the process resides in our heuristic method. Our heuristic method finds the desirability value, or heuristic value, of a node based on the consecutive tokens for each player in a win lane. The method scans through each win lane, and sums the consecutive tokens for each player. For consecutives of 2 tokens, the number is added, unchanged, to the total score for the player. Consecutives of 3 are deemed to be more valuable or desirable than consecutives of two, so they are multiplied by a value globally defined as “MULTIPLIER” before they are added to the total score for the player. Consecutives of 4 constitute a win state, and are the most desirable state. To ensure that these states are deemed the most desirable, they are multiplied by fifty before they are added to the player’s total score for that state. After a total score is reached for both players based on the board state that node represents, the opponent’s score is subtracted from the computer’s score and the resulting number is the final heuristic number for that state. It is important that the opponent’s score was subtracted since it is more desirable for the computer if its opponent’s score is low. The Minimax algorithm uses this heuristic method to obtain heuristic values for each node by choosing either the minimum or maximum value of that node’s children depending on whether that node represents a decision made by the computer of the opponent; the opponent is more likely to choose the minimum value choice and the computer will certainly pick the higher value choice. Eventually these values reach the root node, and since it is the decision of the computer at that point, the root node will take the maximum value of its children and choose the program will choose the column represented by the node with a value equal to the root value—in essence, choosing the child and column with the maximum value and desirability based on that node’s children.

The tree is generated, and the Minimax algorithm’s node values applied, by way of a depth-first limited search. In other words, the Minimax algorithm applies values to the parent nodes as they are created. The search creates each node before travelling to and evaluating it, and then, when the search backs up to the parent nodes, the Minimax algorithm applies a new value to that node based on whether it is greater than its current value (if it is a max node) or less than its current value (if it is a min node). If the node has seven children, is at the depth limit defined by “DEPTHLIMIT” (is a leaf node), or is a win state, then the search backs up to the parent node. When it does that, if the node is a win state or is a leaf node it will first run the heuristic method to give the node a heuristic number; if it is not either of those than the Minimax algorithm has already given it a value.

However, actually committing the computer’s resources to generating all of the possible nodes is wasteful. To avoid creating unnecessary nodes, each node has alpha and beta values that are updated as each of its children are created. If the alpha or beta value is too high or too low (based on whether the node is a min or max node) then the algorithm will “prune” the node and its potential children, since they are unnecessary because their parent won’t get chosen anyway, by ignoring them and moving back up to the parent node right away. Because of this, computer resources are not wasted and the program is able to make an efficient and fairly intelligent decision on which column to pick.

To implement all of this, we chose to use C++ because it has the necessary object-oriented ability to properly implement the tree structure and nodes. Additionally, it is the language with which we are most familiar.

In the course of working on this project, we learned a great deal about the implementation of artificial intelligence methods. When we studied artificial intelligence algorithms like the Alpha-Beta pruning, the Minimax algorithm, and even search methods like limited depth search, we studied them separately. However, in actual practice we learned how these algorithms must work together in order to create the desired effect. While the Minimax algorithm can work without the alpha-beta pruning and still operate, it depends on the depth-first search in order to work.

To simulate an intelligent Connect Four opponent, we represented the board in a two-dimensional array and created a decision tree structure that essentially considers all possible states within a defined ply with nodes containing each possible state. Since creating every single possibility is wasteful of computer resources and takes a long time, we used alpha-beta pruning to limit the amount of nodes that are actually created and considered by ignoring the ones that have no possibility of being chosen by the Minimax algorithm. To generate heuristic values for the leaf nodes, we used a heuristic analysis method that generated a value representing the desirability of that state based on the magnitude of consecutives in each winning lane it contains versus that of the opponent. We used the minimax algorithm in conjunction with a limited depth-first search to find appropriate values for the parent nodes all the way up to the root node, so the program can make an intelligent decision on which column to pick.

References

Russell, Stuart J., and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ: Prentice Hall, 1995. Print.