Tic Tac Toe
Assignment 1
CSC 242

Nick Graham

February 14, 2017

1 Objective

The objective of this assignment was to create a program that would be able to play both 3x3 tic tac toe and 9 board tic tac toe, using a state space search model in order to select its next move.

2 Usage

2.1 Compile

This program comes included with a make file. In order to compile the program please run make. In order to remove the exacutable run `make clean`.
Requirements:

- gcc

2.2 Run

In order to run the code, the format is as follows `./TicTacToe <game>` where `<game>` is either 3 for the 3x3 Tic Tac Toe or 9 for the 9 Board game.
Example: `./TicTacToe 3`

3 Design

The program was written in C due to my familiarity with the language. While C provided me with an easy way to get the project up and running, my limited knowledge made it hard to generalize many of the methods, which resulted in a large amount of repeated code.

3.1 State

In order to store the program state I created 2 structs, one for the 3x3 board, and one for the 9 Board game. The struct containted a char array for the game board, a char for whose turn it currently was, and in the case of the 9 board two integers to store the last played board and spot.
I had originally planed to use an integer to represent the game board, using two bits to represent each square on the board. This idea had been suggested by a TA in order to cut down on memory, however I decided against that in order to keep the implementation simpler.

3.2 Move

I also used a struct to define the moves for both the 3x3 game and the 9 Board game. For the 3x3 game the struct had 2 integers. The first was the spot that was played in, and the second was the cost of the state that results from that action being applied.
For the 9 board move I added and extra integer to store the board being played in.

3.3 Terminal State

To find the terminal states I used function to search the board. I looped through each row and column checking if all the elements were the same, and after I checked the two diagonals. Finally I checked to see if the board was full resulting in a tie.
For the nine board game I creates a separete function that broke the full board into 9 individual 3x3 boards and passed them to the function used before to check and see if there was a winner. It also counted the number of boards that were ties, in order to check and see if all the spots had been filled.

3.4 Valid Move

In order to check if the move was valid, for the 3x3 board I simply check to make sure that the spot was empty.
For the 9 Board game, I applied a similar stategy, however I also included checks to make sure that the board that was being played in corrisponded to the last played spot, unless it was the first move or the designated board was already full.

3.5 minimax

To search the tree I used the recursive version of minimax that was shown in the textbook, *Artificial Intelegence: A Modern Approach.* For both versions, I implemented a max depth, however it was not needed for the 3x3 game, as it was able to search the whole tree fast enough that there was no need to limit the depth. For the nine board minimax implementation I also implemented alpha

beta pruning. I did this because the algorithm was not able to search far enough in the tree in a reasonable amount of time. Without alpha-beta pruning the algorithm was able to search to a depth of 5 before there was a long delay, and with alpha-beta pruning it was able to search to a depth of 13. I then made a gradually increasing depth, so that as we traversed further down the tree there were less valid options, we could travel further down the tree.
Both of the minimax functions relied on both a utility and heuristic function. I combined the two together into one function that would check if the state was terminal and if so return the utility, otherwise it would return the heuristic functions value.

3.6 Utility

For the utility function, I incuded it in my function for checking if the state was terminal. It checked to see who won the game, and if it was the AI, return a value of 1, it was the other player return a value of -1 and if it was a tie, return 0. This was based off of the class discussions on how to implement a 0 sum game from class.

3.7 Heuristic

The heuristic function for the 3x3 board simply returns 0. When I started implementing this I set it to just return 0, however once I found out that I was able to search the whole tree, and did not need the heuristic function, I did not continue to develop it.
For the 9 board game the heuristic function is a bit more involved. The heuristic function for this part searches the boards and penalizes states that have empty boards, or states that have full boards. The goal in this was to to force the moves in a large number of boards in order to drag the game out until a point in which the AI could search the tree to a desirable terminal node. It also checks for boards with 2 pieces in a row or column, as well as giving a weight to a few of the more desirable spots to play in.

4 Results

4.1 3x3 Tic Tac Toe

The results for the 3x3 tic tac to game were favorable. The program was able to search the entire tree from the initail state of an empty board, so the AI never lost, it would always win or tie.
On average it took 0.0020995 ms to search the tree.

4.2 9 Board Tic Tac Toe

The results for the 9 Board tic tac toe were not as desired. The AI unfortunely lost very often because my heuristic was not powerful enough to accuratly pre-

dict the utility of the states. This is an area that I would spend more time on in future attempts, as opped to putting time into speeding up the algorithm. I would like to have searched the board for near wins, or when there were two pieces in a row or column and reward or penalize that as necissary. My attempt to spread peices across the board was also not as affective as I had hoped, and I would like to spend more time working on that to ensure a more even distribution of boards are played in.
On average it took 26.60500646667 ms to search the tree.

References

Artificial Intelegence: A Modern Approach