Assignment 2, Part 1 for CSCI B551

Saurabh Mathur, Shivam Rastogi, Virendra Wali

October 26, 2018

Betsy

Evaluation Function

[['x', '.', '.'],

['x', 'o', 'x'],

['x', 'x', 'o'],

['o', 'o', 'x'],

['x', 'x', 'x'],

['x', 'x', 'x']]

Count the number of continuous pieces a player has like (x, xx, or xxx) in each column and row only for the first n rows. The count can take the values from 1 to N. After counting the values, we will have something like below.

Number of ‘xxx’ in rows = 0

Number of ‘xx’ in rows = 1

Number of ‘x’ in rows = 3

Similarly, we will calculate for the columns and rows. The final value is calculated by using the following formula:-

(Count of sequences)*2^(length of sequence)

For our earlier example,

0*(2^3) + 1*(2^2) + 3*(2^1) = 0+4+6 = 10

Similarly, we are calculating the values for columns and rows of ‘x’. We will also calculate the same values for min player and then subtract the score of min from max.

This evaluation function works amazingly better as it is able to recognize both threats and opportunities for a win.

We also assign high positive and negative values if we find goal states at the shallow depth depending upon max is winning or min is winning.

Evaluation functions we tried

1) Evaluation function 1:

Our first two evaluation functions were naively designed. A brief explanation of these functions is given below:

a. In this evaluation function, we were counting the number of max players and number of min players in each row, column and diagonal, and then we calculated the cost for each row, column and diagonal.
b. After calculating the values, we are subtracting the values for min from max player to check whether the current state is good for max player or min player. If the value is high, the state is good for the max player, and if the value is low the state is good for min player.

For example:

[['x', '.', '.'],

['x', 'o', 'x'],

['x', 'x', 'o'],

['o', 'o', 'x'],

['x', 'x', 'x'],

['x', 'x', 'x']]

Sum of rows for first n rows for both (‘x’, ‘o’) : [(0, 1), (1, 2), (1, 2)]

Combined cost of all rows: (2^2 + 2^2 + 1) - (1 + 1 + 0).

Similarly, we calculated the costs for columns and diagonals and then added them together.

Result: The heuristic did not perform well and was as good as randomly selecting the board. For the bottom 3 rows, it will randomly pick the moves. It also has a considerable number of other flaws like if x dominates in rows(x has densely located in a few rows), and y dominates in columns, then the heuristic will simply remove this information as it adds the scores. There were many other cases and the heuristic failed terribly.

2. We decided to check only goal states in the game tree without any heuristic function. A positive value was assigned to the goal state of max, and a negative to the goal state of min. If no player wins, and we reach the maximum depth then we assigned 0.

Result: It performed better than the first evaluation function but still not smart enough.

Challenges faced

1. We faced some difficulties in understanding whether the evaluation function should take into consideration whose move it is when we are building the game tree. For example, if it is min’s turn should we subtract the score of min from max, or vice versa.

2. We also faced performance issues as we were using a list of lists to represent the board. We eventually decided to change our complete implementation using a single string to implement the board.