Milush Yanev

11/20/2019

CS 4200

Professor Atanasio

Tic Tac Toe

Four in a row

Eight by Eight board

Using Alpha-Beta Pruning

In order to complete this project, we had to follow some guidelines of how the input should be taken and how it should be represented. We also needed to create a function to ask user if he or the agent should start first.

My approach to the project was first to create the game and validation functions for input and output. Even though it sounds easy I ran into a lot of problems implementing the output function. In the beginning I created a 2D board that was displaying just the game input/output but after spending hours to figure it out how to display everything in line by line I concluded that a List will be a good idea. I store every single line of the 8x8 board plus the additional information about user's/agent's move into a List and every time a change is made, I delete the current row and rewrite it with the new information. My program has a total of three Lists build in – one to hold the board itself, one to hold user's moves and lastly one for the agent's moves. The display of is a simple for loop that displays the first eight lines of the three combined Lists and after the eight row is reached, it continues with simply the output of the moves.

One of the issues I ran to was to change the values of 'X' and 'O' respectively to who is starting first. The solution to this problem was solved by passing a value, which indicates who is starting first and from there I created another character variable that will change respectively every time the game is started. This function was used for both user's and agent's moves in order to better and accurately display the game.

The next step was to create a counter that will count the moves of agent and user and increment by one after both made their move. I used the same value passing variable, from which I track when to increment by one and when not to. I also used it in order to determine the accurate display of number of moves and when it's the agent's or user's move.

After trying a lot of different ways of printing I found out that it is best to print the results after every time a faction for user's or agent's move is called. The only time I call just the printing function itself is when the empty board needs to be displayed.

The next step was to create a function that will track for four in a row. This thing was easy, I simply created a Boolean class WinCheck that has held a character value. From there I just look for four matching characters in row or column in our board.

Next step was to create a Boolean called Winner that gets the value of WinCheck and returns true if there is a winner or false if there is isn't. The main idea behind it is to create a do/while loop later and until there is a winner, keep playing.

I needed to create several checks for user's input and to make sure that the input the user is entering is acceptable. There are four steps, first I am checking for a valid format. I am doing this by reading the input characters, in our case two. The first one is to check if the first character is from the range a-h by reading its ASCII representation and the second must be from a range one to eight. The second one is to check if the current place is occupied by checking if the EMPTY character is there. The third one is to check if it is in bounds. The last one is to be used thought out the game and it will check input to be only integers from a certain range.

The next step was to create a min/max functions which I was going to use to evaluate the score for the alpha beta pruning. I found some good references online but for simpler problems. As usual geeksforgeeks had a good example of how the general algorithm works but I already knew that from class. Both of mine min and max functions are identical with just some little changes. Once I found out how to implement them, the rest was easy.

The first issue I ran to was designing the function. I realized that there is no way of creating it without having something to evaluate the scores from the current depth. I created the evaluation function next.

My evaluation function was simple in the beginning but after everything was done, I expanded it a lot. My first approach was to return some score when there is two characters right next to each other. I did it by creating integer points set up to zero and a return of the points as some other value when the board has two characters right next to each other. I just created a nested loop, that loops from row to SIZE and an inner one that loops from column to two. After it returned a result, I knew that I am on the right path.

Next step was to return to min max functions. I created a depth integer set up to a value, then passed integers called alpha and beta. The next step was to read the board. I declared a value for best possible value that will check the maximum of current values in the board. The next step is to set up the function. If there is no winner, the depth is zero or the time is out we return the value from the evaluation function. My depth is set up to four because it takes forever to go to five and it still gives rational decisions at four. Then again, we loop through the board. First, we check if and which positions are empty. Then we initialize them with a character in my case the user's one for our evaluation function to return a score that the function will get. Then I set up my current score which is the minimum one for that current depth using alpha/beta. If the current score is smaller than best score (MAX), set up current=best, else return the best one. For max is opposite – if current is greater than best, set up current=best, else return best.

The most issue I ran to was when evaluating how max/min works and how it reads from evaluation function. I was able to manage it by putting in a lot of hours debugging and reading the data.

The next step was to expand my evaluate function. I did it by printing a sample board, using my pencil and trying possible moves. The most fun part is when you play the game and you see a lot of missed opportunities and you must go back and create another possible move in the evaluation function and assign it with an importance score. The scores are assigned based on importance – if the agents sees and opportunity to block simple move, it's a smaller score but if there is a possible block of winner move by opponent (EMPTY X EMPTY X EMPTY) I give my agent one of the highest scores if it puts "O", on the second "EMPTY" place.

Another issue that I had is when expanding my tree. I am cutting my search with a depth of six which is almost impossible to reach with my current program and especially when the timer is set to eighteen seconds. I found out that if I keep my depth at four my agent works okay. It needs too much time to go to depth five and after depth five is crashing.