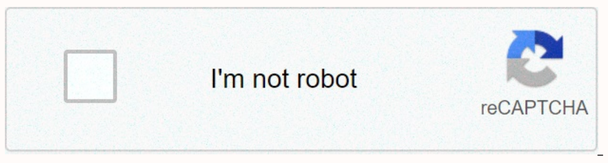

Continue

Game tree for tic tac toe problem

ECS 170- Introduction to Artificial Intelligence; Homework Assignment #2 - Solutions Winter 2003 Rao Vemuri (70 points) - The third problem discussed in the classroom was added at the end of the assignment: 21 January 2003. Regarding: January 28, 2003 (1) (25 points, 5 pts for each part) Consider the Tic-tac-toe game. The game involves playing on the board with 9 squares arranged on a 3 x 3 grid. Two players in turn; one player inserts a token (cross) and the other puts another token (circle) into one of the available empty squares on the game board. The game continues until one of the players receives all three of his chips in a row, column or along the diagonal. a) Estimate the number of states in this game. Not only specify a number. Show me how you got it. At the first level, the branching coefficient = 9. The next level is 8, and so on. So, one way to look at this problem is to say that there are 9! member states. b) Show the entire game tree from an empty board to a depth of 2 (t. y. one X and one O on the board, taking into account symmetry (you should have 3 positions at level 1 and level 12 at level 2) Let's define the canonical positions as 1, 2, 3, 4, 5, 6, 7, 8, 9, going from the upper left square to the lower right square as a wheel. On the first level, one corner (square 1) is a state with x, the middle of which is x (square 5), and in the middle of the side there is x (square 2). Everyone else is symmetrical. On the second level there are 5 children in the corner of the knot with x, these children receive o 2, 3, 5, 6, 9 positions. five kids in the knot with x on the middle side and these kids get o positions 3, 4, 5, 7, 8. only two subdes of the node with x centred. © Now it's easy to calculate their evaluation function using the following formula. (d) Highlight the backup positions of levels 1 and 0 on the tree using the minimumx algorithm and then use them to select the best step. e) Encirse nodes that would not be evaluated at level 2 if alpha-beta pruning were applied, assuming that nodes are generated in the optimal alpha-beta pruning procedure. (2) (25 points, 8/8/9 points) Let's continue the tic-tac-toe problem. It is now in our interest to look at the programming aspects of this problem. One of the first issues that we need to address is the issue of representation. Let us do a few attempts towards this goal. (a) First test: TTT1: Define the data structure as shown below. Data maps: 1) Nine-element vector (array) depicting the board · | PART 1: PART 2: 3 |· | 1 December 2004 5 6 December 2004 |· | December 7, 2004 December 8, 2004 1 December 2004 Each item contains values 0 === square is empty 1 === square has X 2 === square has O 2) Movable: 3^9 element vector (19 683). Each element is a vector of nine elements. Algorithm: 1) Use the current board status specified by the ternary nine-digit number convert to decimal 2) Use the number from top as index to move and reach the vector stored there. 3) Action 2 vector is the correct step. It indicates what the board will look like after the transfer. Set the board equal to vector comments: Advantages: 1) effective time 2) optimal theoretically Disadvantages: 1) a lot of space (too much?) 2) Someone has to specify all 3 ^9 entries (19683 entries) 3) errors indicating all 3 ^9 entries easy to do 4) Extension? (3 dim 3^{27} moving records, 4x4....) b) Second test: TTT2: Let's define the data structure as shown below. Data structures: 1) Board: nine-element vector representing the board · value 2 === empty· value 3 === X· value 5 === O· Why?... 2) Turn: integers indicating which movement · turn = 1 first step· turn = 9 last step Algorithm: Three procedures. 1) Make2: Try to make two in a row. He first tries to play in the center, then tries noncorner squares. 2) Posswin(p): · If the player p can not win the next step · Go back 0· a.m. Returns the square that is· is a winning move We first call posswin (us) if we can win we win the move. If we don't win we call posswin (opponent) and block opponents from moving if he/she can win. posswin checks each row, column and diagonal If (product == 18) X can win ((3 * 3 * 2) = 18) If (product == 50) O can win ((5 * 5 * 2) = 50) This means to scan the row / column / diagonal to find the blank space you want to move to. 3) Go (n): Go to square n X starts! Algorithm details: 1) turn = 1 go(1) /*upper left angle*/ 2) turn = 2 · if (board[5] empty) · go(5) · still · go(1) · if(the board[9] is empty)· go(9)· a.m. go(3) · if(posswin(X) != 0)· go(posswin(X)) /*block opponent*/· a.m. go(make2) Comments: Cons: 1) Not so effective time 2) The overall strategy was the numbers of the pre-programmer. Policy errors? 3) Do not summarize the advantages of up to 3-D or other games: 1) Effective space 2) Time is not so bad 3) Easy to understand strategy or change it. (c) Third test: TTT3: Propose an alternative wording and alternative algorithm (simply specify the algorithm in a few steps, at each step in unofficial English) and its pros and cons. Decision data structures: 1) Position of the Board: Structure containing - nine elements vector representing the board - List of board positions that may arise as a result of the next step. - a number indicating /evaluating the probability of each step winning Algorithm: 0) consists of a list of subsequent board positions 1) Look at the list of subsequent board positions 2) decide which one is best, make a move and pass back this best moves digital rating Judging which step is best: For each step 1) See if you can is to win if so give it the highest ranking 2) If you don't win: Consider all the moves an opponent could make another watch, which one is the worst for us (recursive call) Let's say that the opponent will make this move (What the worst for us is the best opponent) to return this ranking 3) the best step step with the highest rating (3) (20 points, you will lose one point for each error) in the classroom, I drew a tree on the board and showed how to do alpha-beta pruning. Complete this procedure for the remaining tree 000 000000 and show the best sequence of maximumr motion of the player. In today's article, I will show you how to create an unbeatable DI agent who plays the classic Tic Tac Toe game. You will learn the concept of the Minimumx algorithm, which is widely and successfully used in areas such as artificial intelligence, economics, game theory, statistics or even Philosophy.AI X and Human OTable of Contents Before going to part of AI, let's make sure we understand the game. About Tic Tac ToeTic-tac-toe (also known as noughts and crosses or Xs and Os) is a paper and pencil game for two players, X and O, who take turns marking spaces on the 3×3 grid. A player who manages to put three of his marks on a horizontal, vertical or diagonal line wins the game. I recommend you play the game yourself, feel free to check out your iOS Tic Tac Toe app. In order to sove tic tac toe, we have to go deeper than just thinking about it as a game where two players place X and O on the board. Formally speaking, Tic Tac Toe is a zero-sum and great informational game. This means that each participant's profit is equal to the losses of other players and we know everything about the current state of the game. In a two-player (A vs B) game, if player A scores x points (utility units), player B loses x points. Total profit/loss will always fall to 0.With this in mind, let's move to the Minimumx algorithm that is suitable for such cases. Minimumx algorithmMinimax is a recursive algorithm used to choose the optimal player step assuming that the opponent also plays optimally. As its name suggests, its goal is to reduce the maximum loss (reduce the worst case scenario). You can think of the algorithm as a representation of the human thought process by saying, OK, if I make this move, then my opponent can only make two moves, and each of them would allow me to win. So this is the right move. Let's look at the code! Line 1 To calculate the minimumx score, let's feed the minimumx function in the current board status ([Player]) and the opponent player (Player)Lines 2-4Then to see if the board already has a winner. If it's a player who's been transferred to a function, we're going back 1, otherwise -1.Lines 9-19After, let's try all possible movements and recursively calculate him a minimumx score. If the minimumx function does not find the terminal state, it keeps recursively going level deeper into the game. This recursion occurs until it reaches the terminal state and returns the result one level up. 20-23 rows If the score has been updated, we return it as a minimumx score; otherwise, it's a draw that we're coming back 0.I know. At first, it may sound frightening rather than intuitive. on its recursive recursive but when you see it in action, you'll easily understand. Without further noise, let's go for an example to better understand what's going on. Unparalleled Tic Tac Toe AILet look at this state of the board. We will try to fix it as X.How do you solve it? I think you would think that the best move in this case would be to place x in the middle of the board and win the game. And you would be right, but is it the only win decision X? How do I get this solution? To determine this, let's draw a tree for all possible board states. Look for the tree's 0.0 state and X turn You can see above, starting with the original state of 0.0, we have 3 options (1.0, 1.1, 1.2).1.0 gives us a victory (+1), but let's explore other paths.1.1 gives our opponent 2 options (2.0, 2.1). 2.0 is a winning state for our opponent, so for us it is a lost state (-1). 2.1 gives only one chance 3.0 in which we win (+1).1.2 gives our opponent 2 chances (2.2, 2.3). 2.2 is a winning state for our opponent, so for us it is a lost state (-1). 2.3 gives only one chance 3.1, in which we win (+1). Okay, but how can we interpret that? Let's start with the terminal states at the bottom and calculate minimumx scores. At depth 3 we maximize, so we multiply +1 points into previous movements in depth 2.At depth 2 we minimize, so we multiply -1 points in the previous movements in depth 1.At depth 1 we maximize, so we multiply +1 to the previous step in depth 0.Finally in depth 0. where we really are, we have to choose to move related to the +1 result we end up with. Tic Tac Toe AI decides to go to 1.0 knot and win the game. Our Tic Tac Toe AI performs such simulations for every move, making it the most unparalleled opponent. But what makes it unbeatable? Due to the relatively small state space (3^9 = 196839 possible combinations of boards), it can easily search the whole game tree for optimal solution, treating the game as a completely deterministic environment. On the other hand, chess, for example, has an extraordinarily large state space ~ 10^{12}民 opportunities. This is a much larger number than the number of atoms in the observable universe. With such extensive search spaces, we can still use the Minimumx algorithm, but we must keep in mind limiting the depth of our search engines. Otherwise, we can finish counting results for a very long time or even forever. In short, the smaller the state space, the better results we can achieve using the Minimumx algorithm.tl;Dr. Tic Tac Toe AI is unparalleled. You can draw the most and only with a great game. If you think you can surpass it and win the game, let me know how to do it in the comments section😉. If you're wondering about other Tic Tac Toe based games, feel free to check out Achi! This is an extension of tic tac toe that allows you to move the tiles. This subtle addition greatly complicates the and makes it much harder! What's next? Until now, you should be able to understand the principles of the Minimumx algorithm, which allowed us to create an unparalleled agent of Tic Tac Toe AI. Minimumx is a very powerful and versatile algorithm that can be applied in various applications. Do not hesitate to apply it to other problems. Not hoping to see your results! Don't forget to check the project's iOS app. Questions? Comments? Feel free to leave your feedback in the comments section or contact me directly remember to contact 💸 if you liked this article ☺. ☺.