ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

20127110 – PHAN HỮU ĐOÀN ANH

20127512 – TRẦN QUỐC HƯNG

20127643 - TRƯƠNG GIA TIẾN

PROJECT 1 – SEARCH - II

Playing game with adversarial search

| Giáo viên hướng dẫn |

TS. Nguyễn Hải Minh

TS. Nguyễn Ngọc Thảo

Thầy Nguyễn Thái Vũ

Môn: Cơ sở trí tuệ nhân tạo

I. TABLE OF CONTENTS

II. INTRODUCTION ABOUT ALPHA BETA SEARCH

The Alpha Beta Search is an optimal update of the Minimax Algorithm. The same as Minimax, this is an antagonistic recursive function with two-player as MAX and MIN. Two-player will play alternatively with MAX as the maximum node and MIN as the minimum node. The algorithm will return the optimal node based on the MIN-MAX above.

The difference between Alpha Beta Search and The Minimax is that Alpha Beta can avoid or cross out the branches and the neighbor nodes that do not meet the condition. So, The Tree-Search's memory is less Minimax, and we do not have to access all of the Tree-Search nodes.

III. IDEA & DESCRIPTION IDEA

1. Common Idea

This program uses Alpha Beta Search to find the optimal way for BOT to win against Humans as player-1 or player-2.

We create a BOT to test all the empty blocks of the chessboard and find out which is the optimal way to win a player.

Every single move of BOT and player will be counted as one state, every single state corresponding to the return value (Win-Draw-Lose). We use Alpha-Beta Search to recursive state till the final state of every state above to find the value of the state. The recursion is the min function called the max function and vice versa, so MIN MAX is the same as the two players. The Final State is the Draw Case (the chessboard has no more empty blocks), or One of the players is the winner. Then 3 cases above will return three different values, and the optimal path is the maximum value (BOT wins).

2. Problems

The Alpha-Beta Search works very well with a 3x3 chessboard, but when we upgrade to a 5x5 chessboard (having 3^{25} states) and so on 7x7 chessboard (having 3^{49} states). The Tree-Search memory becomes enormous, and this cause the running time will be very long. So the algorithm will have one more parameter: depth (The depth of Tree-Search that the algorithm can go).

In a 3x3 chessboard, the depth of Tree-Search is not necessary. However, it is essential in 5x5 and 7x7. For example, with a 3x3 player 1, go first, and we have 3^8 (6561 states) left, but in 5x5, the number will be 5^{24} = 59,604,644,775,390,625, and with 7x7, the number will be more greater than the above. To decrease running time, determining the depth for the algorithm to search is necessary.

If the depth is too tiny, the algorithm cannot work optimally because the algorithm still cannot find the correct state. Furthermore, The BOT will choose the wrong way and lead to losing. Besides, if the depth is too high, the time thinking of the BOT is higher.

After several tests with 5x5 and 7x7, we can see the correct depth for 2 cases to decrease time thinking and win the human.

IV. DESCRIPTION CODING

- Using pygame to create UI
- Entire chessboard will be saved as 2D-matrix and every movement will be in this chessboard.
- The program has a main function to call all of the functions.
- Processing of Algorithm:
    - Alpha_beta_search // call min_value
    - Min_value // Find Minimum value of algorithm
    - Max_value // Find Maximum value of algorithm
    - Cal_state_value // Calculate the current state
    - isEmptyTable // Check if the table is Empty or Not
    - Best_move // Find the largest moment of algorithm search
- The rest of the function is used to create UI.

V. PSEUDOCODE ALPHA BETA PRUNE FOR TIC TAC TOE

Using pseudocode from slide to create the algorithm, The Alpha-Beta Prune will have three functions name alpha_beta_search, max_value, min_value, and best_move to call alpha_beta_search and find depth for 3x3, 5x5, and 7x7. Two subfunctions is Cal_state_value and isEmptyTable.

Cal_state_value will calculate the cases the player or bot will win with row and column, with 7x7 being the primary cross, secondary cross, and other crosses.

Function alpha_beta_Search(board, depth)

Return min_value(board, depth, alpha, beta)

Function max_value(board, depth, alpha, beta, tranpositiontable)

Calculate the current state, if the player win return 1, unless return -1 and if the table is full return 0

If depth = 0 return value of current state.

Best = -inf

With every position in table

Try and find the best value (best) best = max(best, min_value(depth -1))

If best >= beta cross out (prune) → return best

If not, alpha = max(alpha, best)

Function min_value(board, depth, alpha, beta, tranpositiontable)

Calculate the current state, if the player win return 1, unless return -1 and if the table is full return 0

If depth = 0 return value of current state.

Best = inf

With every position in table

Try and find the best value (best) best = max(best, min_value(depth -1))

If best <= alpha cross out (prune) → return best

If not, beta = min(beta, best)

Function best_move(board, count)

bestValue = -Inf, bestMove = (-1, -1)

Depth = size of chessboard.

With every empty block of table

assign the current block with value and sign of BOT

Using alpha beta search to find the best new value (new) for current state

If the best new value (new) > the current best value

Assign best move to the current position

Assign best new value to the best current value (new)

Assign the best move for bot

VI. COMPLETENESS

Minimax is a search algorithm with completeness because of its search tree, although many still have only finite steps, and the alpha-beta search algorithm also inherits this property of Minimax. The algorithm can always return an outcome that is the best move if the move exists (i.e., the game ended with a player winning or is still empty).

VII. TIME / SPACE COMPLEXITY

1. Time complexity

The complexity of alpha beta prune:

Worst case: $O(b^d)$ This case is our alpha-beta search turning into normal Minimax, which is impossible to prune any branches, traversing all branches of the search tree.

Best case: $O(b^{d/2})$ The case that we call "good ordering" it easily prune any branches. The above also means that, with the same amount of time, alpha-beta prune can search to a deeper depth than Minimax, and if the same depth, alpha-beta prune will be faster than Minimax.

2. Space complexity

Space complexity of alpha beta search is O(bd) corresponding to b is the branching factor, and d is the max depth of the algorithm.

VIII. DEMO LINK

Onedrive: tictactoe.mp4

IX. REFERENCES

ADVERSARIAL SEARCH

1. https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning
2. https://www.youtube.com/watch?v=N98F8HYEDCk
3. https://www.youtube.com/watch?v=trKjYdBASyQ
4. https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-3-tic-tac-toe-ai-finding-optimal-move/
5. https://stackabuse.com/minimax-and-alpha-beta-pruning-in-python/
6. https://ai.stackexchange.com/questions/5174/what-else-can-boost-iterative-deepening-with-alpha-beta-pruning
7. https://homepages.cwi.nl/~paulk/theses/Carolus.pdf
8. http://www.cs.utsa.edu/~bylander/cs5233/a-b-analysis.pdf
9. Powerpoint slides in theory class

PYGAME

10. https://www.pygame.org/docs/
11. https://www.geeksforgeeks.org/how-to-create-buttons-in-a-game-using-pygame/
12. https://stackoverflow.com/questions/52710092/loop-within-main-loop-in-pygame
13. https://eng.libretexts.org/Bookshelves/Computer_Science/Programming_Languages/Book%3A_Making_Games_with_Python_and_Pygame_(Sweigart)/03%3A_Pygame_Basics/3.04%3A_Game_Loops_and_Game_States