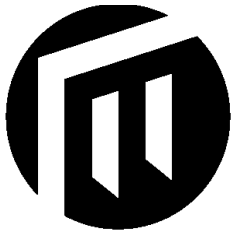

Alpha-beta pruning Othello

Artificial Intelligence assignment 1

Linus Granath

Table of contents

1 Introduction

The goal of this assignment was to implement the alpha-beta pruning algorithm in order to create an agent which can play Othello, in this report I will explain how alpha-beta pruning was implemented, how the application works and which maximum depth is optimal.

2 Application structure

The application consists of seven classes, as seen in Fig. 1. In this chapter, each class will be described in more detail.

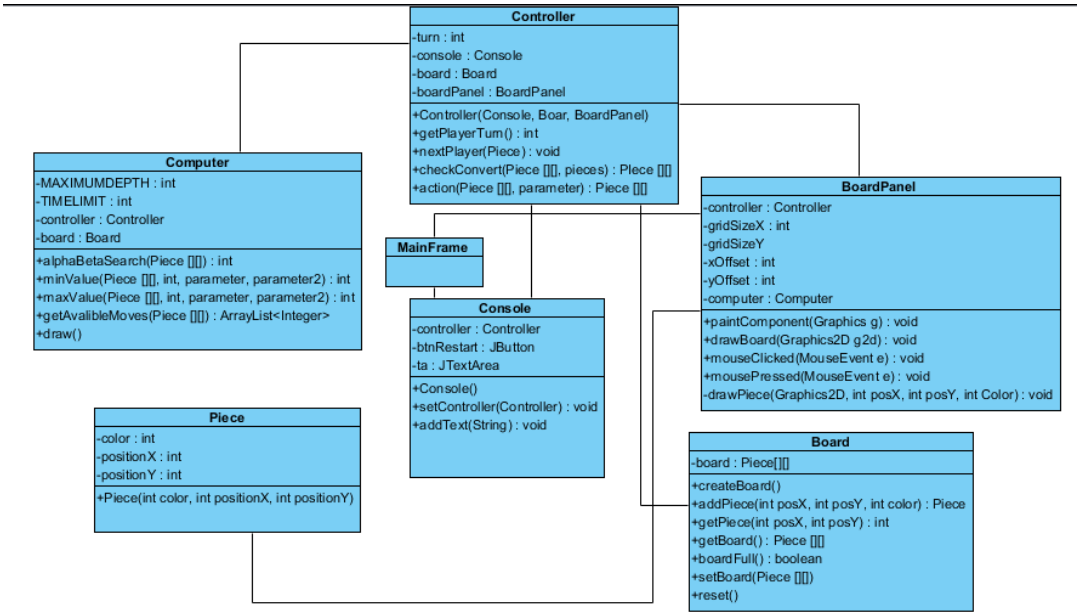

Fig. 1 Class diagram of the application

2.1 MainFrame

The purpose of the MainFrame class is to start the game and create the window that the game will be displayed in. The MainFrame also creates a BoardPanel and Console object which are added to the frame.

2.2 Console

The Console displays useful information in a JTextArea, such as the amount of nodes searched, and how deep the search went in the alpha-beta pruning algorithm.

2.3 Piece

The Piece class contains an x-position and a y-position, indicating the position of the piece on the board. It also contains a color, black or white which indicates which player controls it.

2.4 Board

The Board class contains the state of the game, represented by a 2-dimension array containing Piece objects. The Board also contains a function to manipulate the state by adding pieces to it.

2.5 BoardPanel

The purpose of the BoardPanel class is to display the state of the game by drawing the board and the pieces. It also handles input from the mouse, which is used to determine where the player wants to place his pieces.

2.6 Controller

The controller handles a lot of the logic of the game, such as converting pieces when a move is made and deciding the player turn.

2.7 Computer

The Computer class is the agent using the alpha-beta pruning algorithm. The algorithm is used to decide the best possible move for the computer. It contains functions to use the algorithm and to execute the best action.

3 Search Algorithm

The search algorithm was based upon alpha-beta pruning, by using the skeleton provided at page 170 in the book [1]. The algorithm works with three functions, alphaBetaSearch(), maxValue(), minValue(), which will be described in more detail below.

3.1 alphaBetaSearch

The alphaBetaSearch function takes the current state of the gameboard as input, i.e the position of all the pieces on the board. The main purpose of the function is to find the best possible move within five seconds.

At the start of the function, we store the current time, so we can keep track on how long the search has been going on. We also reset the depth and nodesexamined variables, which tells u show far down the search we've gone and how many nodes that have been searched.

The best move is retrieved by calling the maxValue function, with the arguments: Current state, depth set to 0, alpha set to the smallest possible int number, and beta to the biggest possible int number, we also send the start time. The last thing the function does is to display how many nodes we searched, how deep and which action was made. The function retrieves the move which yields the highest utility and returns it.

3.2 minValue

The **minValue** functions takes the state of the gameboard, the current depth in the tree, the alpha and beta values and the time at the start of the search. The main purpose of the **minValue** function is to make the best possible move for the player.

3.3 maxValue

The **maxvalue** function takes the state of the gameboard, the current depth, the alpha and beta values and the time at the start of the search. The main purpose of the **maxValue** function is to make the best possible move for the computer. The function is recursive and simulates all the available moves and retrieves the best possible move. The best move for the max player is the move which generates the highest utility. If the depth is equal to the allowed depth, or if there are no available moves the function will return the current utility of the board.

3.4 avaliableMoves

The purpose of the freePlaces function is to return the possible actions left for the players to make. It works by iterating through the state of the board and storing all the empty spaces into an arrayList.

3.4.1 cutOff

When the search for the best move has taken five seconds, the search is interrupted. It also occurs if the computer has predicted the maximum allowed moves, which is 14.

4 Result

In this chapter the time taken for the search algorithm for each depth limits is displayed in graphs. With the time limit of five seconds the optimal depth would be 12, as can be seen in *Fig 2*.

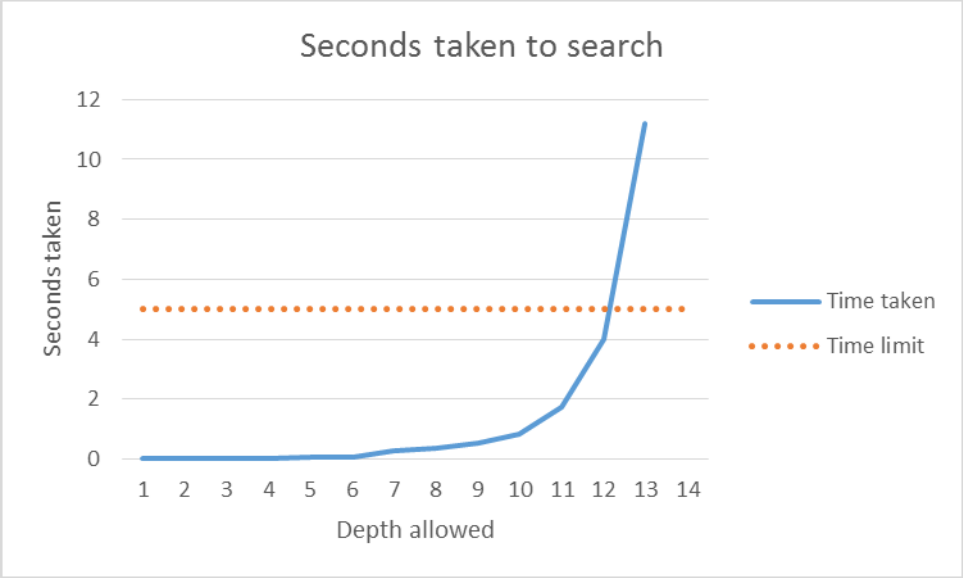

Fig 2. Graph showing seconds taken to retrieve the best action, with the allowed depth.

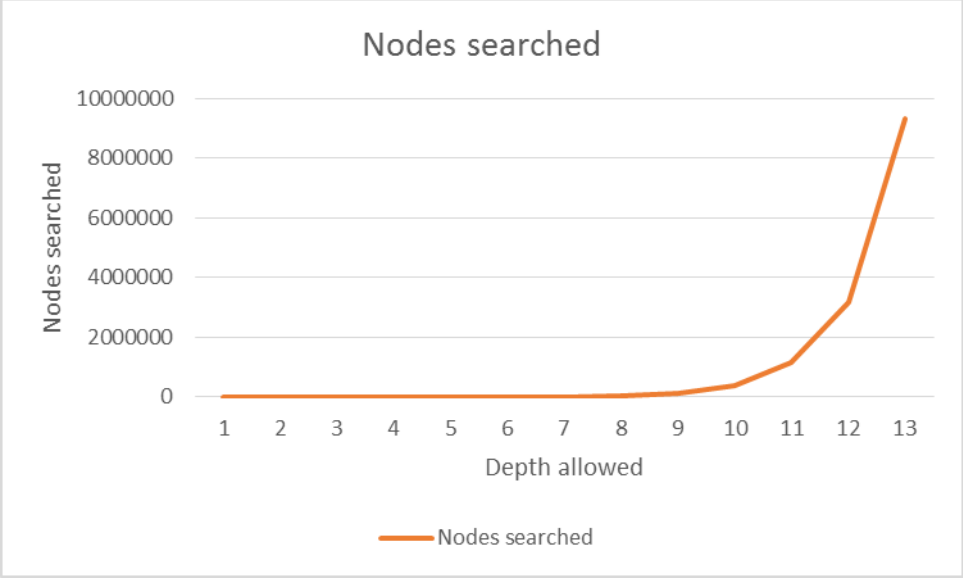

Fig 3. Graph showing the total amount of nodes searched through, with the allowed depth.

5 Reflection

In this chapter the end result of the application is discussed.

5.1.1 Conclusion

The difficulty of the opponent varies, depending on what the maximum depth allowed is, and how fast the computer can search. On this computer the optimal depth would be 12, because the time limit is five seconds, and it takes roughly four seconds for the algorithm to finish, as can be seen in *Fig 2*. If the depth limit was higher, the agent would not be able to find the best action for every state because it would not have the time to search through all of the outcomes. When I set the depth to 14 I could beat the computer by placing my figure in a spot where the agent doesn't have time to calculate a good move. The agent works fine even from the depth 5, but an agent which can predict more outcomes is better.

5.1.2 Reflection

The assignment was both fun and challenging, by using the alpha-beta pruning algorithm I will now be able to create intelligent agents in other games, like Tic-Tac-Toe. I feel like the hardest part about the assignment was to create all of the functions the algorithm is based upon. Such as converting pieces and simulate all of the moves. The algorithm itself wasn't that hard to understand and implement. In the future, I will optimize my minValue and maxValue functions by combining them into one function and change the function so it also takes the current player's turn as input.

Reference list

[1] Norvig, Peter & Russel, Stuart (2009). *Artificial Intelligence: A Modern Approach Third Edition*.