Game of Othello - Using Mini-Max algorithm

AI For Search Methods - CS6380
Ankit Thawal - CS21M004

November 28, 2021

Abstract

1 Introduction to Game of Othello

Othello(Reversi) is a strategy board game for two players, played on an 8×8 uncheckered board. It was invented in 1883. Othello, a variant with a change to the board's initial setup, was patented in 1971.

2 Some examples to get started

2.1 The role of AI

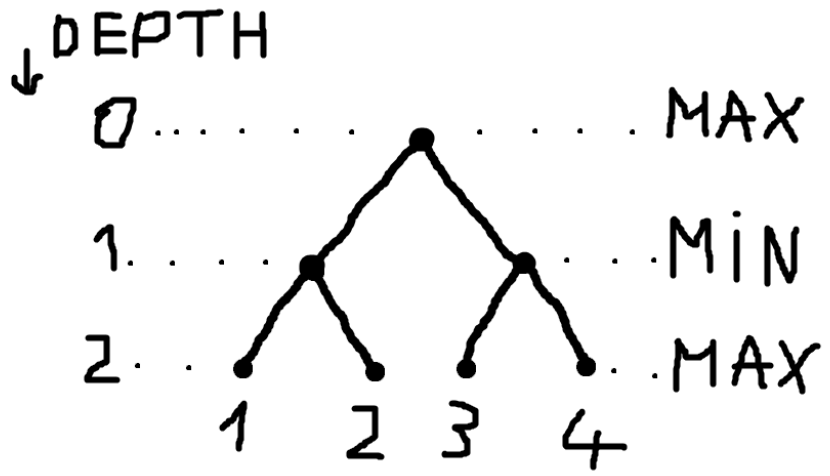
The technology was created to help and facilitate human activities and work. One technology that is being intensively created to create sophisticated devices is Artificial Intelligence. Many artificial intelligence developers who create implementations of artificial intelligence that are super sophisticated and applied to the real world. Artificial intelligence has the opportunity and potential to be developed further with qualified resources.
AI continues to evolve to benefit many different industries. Machines are transferred using an interdisciplinary approach based on mathematics, computer science, linguistics, psychology, and more.

2.2 Mini-Max Algorithm

Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory. It provides an optimal move for the player assuming that opponent is also playing optimally. Mini-Max algorithm uses recursion to search through the game-tree. Min-Max algorithm is mostly used for game playing in AI. Such as Chess, Checkers, tic-tac-toe, go, and various tow-players game. This Algorithm computes the minimax decision for the current state. In this algorithm two players play the game, one is called MAX and other is called MIN. Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit. Both Players of the game are opponent of each other, where MAX will select the maximized value and MIN will select the minimized value. The minimax algorithm performs a depth-first search algorithm for the exploration of the complete game tree. The minimax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.

2.3 Methodology

If we think of a game in terms of these 2 players, Max Min, changing turns with each other, then we can represent the game as a tree of decisions. Let's look at a quite simple example:



Each node in this tree (except for the terminal nodes) represents a decision that should be made at that moment in the game. We decide which move to take. In this example, we have to choose from only 2 movements, but in general, we can have any number of movements, and this number can vary from node to node, depending on the state of the game. The top node (the one at depth 0) is the current state of the game. Here is where we have to decide the next movement in the game. Here we start with Max, as it is the player that wants what we do: to maximize the score. Instead of deciding based only on the next possible states of the game that Max can reach from this point, he thinks: "After I will do one of these moves, what will do my enemy Min?" So, it calls Min and says: "Hey, what move will you take if I choose left?", and after that: "What move will you take if I choose right?". After Max finds what Min would do, he chooses the branch that will give him the maximum score. But wait. How will Min decide what to do to minimize the score? He will apply the same strategy as Max. Min in turn will call Max to ask him what he will do for each possible choice of Min. But after that, instead of making the choice that maximizes the score, Min will do the opposite: will make the choice that minimizes the score. And so on ... Each one will call the other, building a big tree in a recursive fashion like this until they reach a terminal state. A terminal state should be, preferably, a state in which the game is ended. But that's usually too expensive computationally; letting the algorithm explore all the possible moves until the game ends may take an extremely long time. So, we set a maximum depth. A game state will be considered terminal when either the game is over, or the maximum depth is reached. In the example above, the terminal states are the ones on the bottom (depth 2). What will happen on the terminal nodes? Whoever player will reach terminal states, be it Max or Min, will not be able to apply the same strategy used so far; the strategy of calling the other player for help. Now, in terminal nodes, we need to calculate the score of the game in each one of these terminal states. This may not be so obvious in some games, but we need to, at least estimate the score based on that state of the game. In our example above, the scores of the terminal states are those numbers on the bottom: 1, 2, 3, and 4. The only thing that the Max player will do at this last level is to return these scores to the calling Min on the previous level.

```
function maximize(state):

    if is_terminal(state):
        return (NULL, eval(state))
    (max_state, max_score) = (NULL, −∞)

    for child in state.children():
        (_, score) = minimize(child)
        if score > max_score:
            (max_state, max_score) = (child, score)

    return (max_state, max_score)
```

MiniMax with alpha-beta prunig

Alpha-beta pruning is a strategy that we can use to improve the Minimax algorithm by ignoring some branches of the tree that we know ahead of time they won't help us in making the optimal decision. The name Alpha-beta pruning comes from the 2 parameters used in this algorithm which are alpha and beta.

References

1. First course on Artificial Intelligence - Prof. Deepak Khemani