

CPCS331 – Artificial Intelligence – Spring2020 -Project I

[ Using Alpha Beta, Minimax and MCTS algorithms]

|  |  |  |
| --- | --- | --- |
| Student Name | Student Number | Section |
| Elham Imdad | 1779096 | AA |
| Ghadeer Qalas | 1778773 |
| Lena abdulmanan | 1705454 |

# Part 1

# 1. Minimax and alpha-beta pruning technique to solve “Dots and Boxes” puzzle

## 1.1 Introduction

The problem that we’re trying to solve is Dots and Boxes problem. Dots and Boxes is a combinatorial game popular among children and adults around the world. In a game of Dots and Boxes, the players draw a rectangular grid of dots and take turns drawing lines between pairs of horizontally or vertically adjacent dots, forming boxes. A game’s size is defined in terms of the number of boxes. The player who draws the fourth line of a box captures the box. When this happens, the player gains a point and then must draw another line. At the end of the game, the player with the most points wins. If both players captured the same amount of boxes, the match is considered as a tie.

## 1.2 Problem formulation as a search problem

## 1.3 Technical discussion

The AI Techniques for solving " Dots and Boxes" Problem used in this report are:

* **MiniMax technique:**

MiniMax algorithm is the most common artificial intelligence algorithm for a computer to participate in two-player combinatorial games.

It works by looking a head from the current game configuration a certain number of moves to see all the possible states the game could reach, building a game tree that is complete to a certain depth. Each state of the game will contain a score, given by an evaluation function. In Dots and Boxes game, this evaluation assigns a good score to a move that will give points to the player. On the other hand, it will give a bad score to a move which allows the opponent to ﬁnish the square in the future. We`re going to implement this method using java.

* **Alpha-beta pruning technique:**

The Mininmax pseudocode in our problem:

**function** minimax(board, depth, player)

**If** board.depth >= depth or children.isEmpty() **then**

**return** the heuristic value of board

**If** player = "MAX" **then**

value = -∞

**for each** child of board **do**

minMaxVal = minimax(child, depth-1, "MIN")

**If** minMaxVal > value **then**

value = minMaxVal

**return** value;

**else if** player = "MIN" **then**

value = ∞

**for each** child of board **do**

minMaxVal = minimax(child, depth-1, "MAX")

**If** minMaxVal < value **then**

value = minMaxVal

**return** value;

**return 0**

## 1.4 Discussion of results

## 1.5 Results

Minimax results:

A screenshot of a social media post

Description automatically generated

# Part 2

# 2. Monte Carlo Tree Search technique to solve “Dots and Boxes” puzzle

## 2.1 MCTS Algorithm explanation.

A tree structure is a hierarchy of linked nodes where each node represents a particular state. The structure has nodes, these nodes have none, one or more child nodes. There is a particular way for a solution, The exits path from the "root" node (initial state) to a "goal" node (desired state). Tree search algorithms attempt to find a solution by traversing the tree structure, it's starting at the root node and thoughtfully expanding the child nodes in a specific way.

Monte Carlo algorithm is a tree search algorithm that starts from any state and tries to improve it by producing its successors and choose the one that is more optimal than the current node and the other successors. It becomes useful as it continues to evaluate other alternatives periodically during the learning phase by executing them, instead of the current perceived optimal strategy. The process of Monte Carlo Tree Search can be broken down into four distinct steps, viz., selection, expansion, simulation, and backpropagation. Figure (1) shows each of these steps below:

A close up of a device

Description automatically generated

2.2 Example

We will apply the MCTS algorithm for Dots and boxes problem. The dots and boxes game is the problem of starts each player takes turns adding a single vertical or horizontal line between two enjoined adjacent dots. When a player completes the fourth side of a 1×1 box earns one point and takes another turn. The game ends when no more lines can be placed.

In our example, we will play between two humans or between human and computer.

2.2.1 Color of players:

Player #1: holder blue color.

Player #1: holder red color.



2.2.2 Board of game:

The real board representation:

A screenshot of a cell phone

Description automatically generated

The board size is 5X5. On the start of a Dot and Boxes, the blue side is always the first player. The UI will count points and time automatically.

* New Game button to start a new game with the computer.
* To stop the game, hit the Stop Game button on the left side.
* If you want to play with other players, you can Un-lead Engine, start a new game without game engine loaded.

2.2.3 MCTS Algorithm

A screenshot of a cell phone

Description automatically generated

Here where using MCTS algorithm to implements the game, first of all create root and the children of nodes. Choosing the successor depends on its heuristic function. It must be better than the current state and other successors.

The heuristic function is computed by the total number of boxes and lines. This piece of code calculates the heuristic function:

A screen shot of a computer

Description automatically generated

After that judge between two players depends on heuristic function:

A screenshot of a cell phone

Description automatically generated

2.2.4 Final output:

A screenshot of a cell phone

Description automatically generated

## 2.3 Hill-climbing Algorithm Evaluation

Thursday

|  |  |
| --- | --- |
|  | MCTS performance |
| Completeness | No, it might fail when it gets stuck on local maxima. That is, all successors of the current state are worse than it. |
| Optimality | No, but it finds the local optimal solution (a solution which all its successors have higher cost than it, but it costs more than the optimal solution). |
| Time complexity |  |
| Space complexity |  |