VNUHCM - University of Science

The Faculty of Information Technology

PROJECT REPORT 2:

THE ADVERSARIAL SEARCHING ALGORITHM

Subject: Fundamentals of Artificial Intelligence

Supervisor : Pham Trong Nghia

Nguyen Thai Vu

Student's Fullname : Thai Chi Hien

Student's ID : 20127495

TP HCM - 2022

Table of Contents

1. Project Requirements:

- In this project, students research and implement the adversarial searching algorithm.

- In addition, students implement an application (tic-tac-toe problem) and apply the adversarial technique to solve that tic-tac-toe.

- Students implement a tic-tac-toe game (Vietnamese: trò chơi caro). For simplicity, students just need to implement 3x3 and 5x5 maps. Student will control player 1, Computer will control player 2 (and vice versa).
- **Students choose any adversarial search. Using that adversarial search to find the optimal path, which will help the Computer to win this game.**
- Students must implement the interface of tic-tac-toe game. Note:
  - You can use pygame or tkinter (python library), those libraries are so easy to learn and use.
  - The main purpose of this project is learning Adversarial search, please do not focus on application or interface (just easy to look are enough)

2. Project Progress:

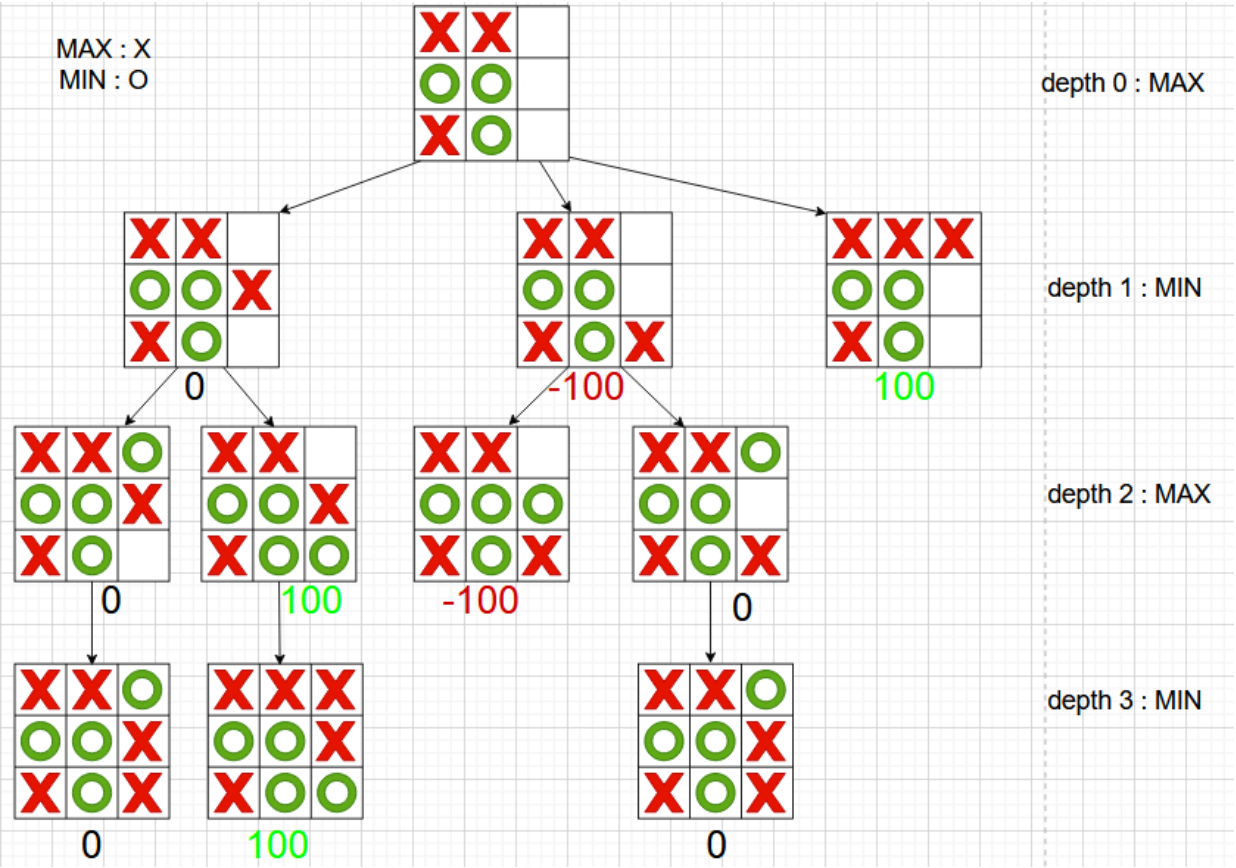
#	Task	Note	Progress
1	Implement a tic-tac-toe game	Both players (human and bot) can play tic tac toe against each other	100%
2	Implement a computer can play 3x3 tic tac toe map	Bot will try to win or draw	100%
3	Implement a computer can play 5x5 tic tac toe map	Bot will try to win or draw	100%
4	User interface to be able to select a map and play	Can choose 3x3 or 5x5 map and an option to play again	100%

3. The Aversarial Searching Algorithm:

a. Mini-Max Algorithm:

- Mini-Max is a recursive or backtracking algorithm that is used in decision making and game theory. It will try to find a move to *minimize* the maximum possibility to lose the game and also *maximize* opportunity to win the game. Mini-Max algorithm is commonly used in turn-based two-player games such as chess, tic tac toe, go,… where the algorithm considers one player as MAX and the other as MIN. MAX will choose the move with the greatest score while MIN try to find the move with the smallest possible score.

- Mini-Max algorithm usually represented as a complete game tree with each node is a possible move and the root node is the current state of the game. The algorithm uses depth-first search to discover and reaches the leaf node to calculate the value, then backtrack to find value of the upper-node. The node representing the MAX will contain the maximum value of the child nodes while the the node representing the MIN will contain the minimum value of the child nodes.



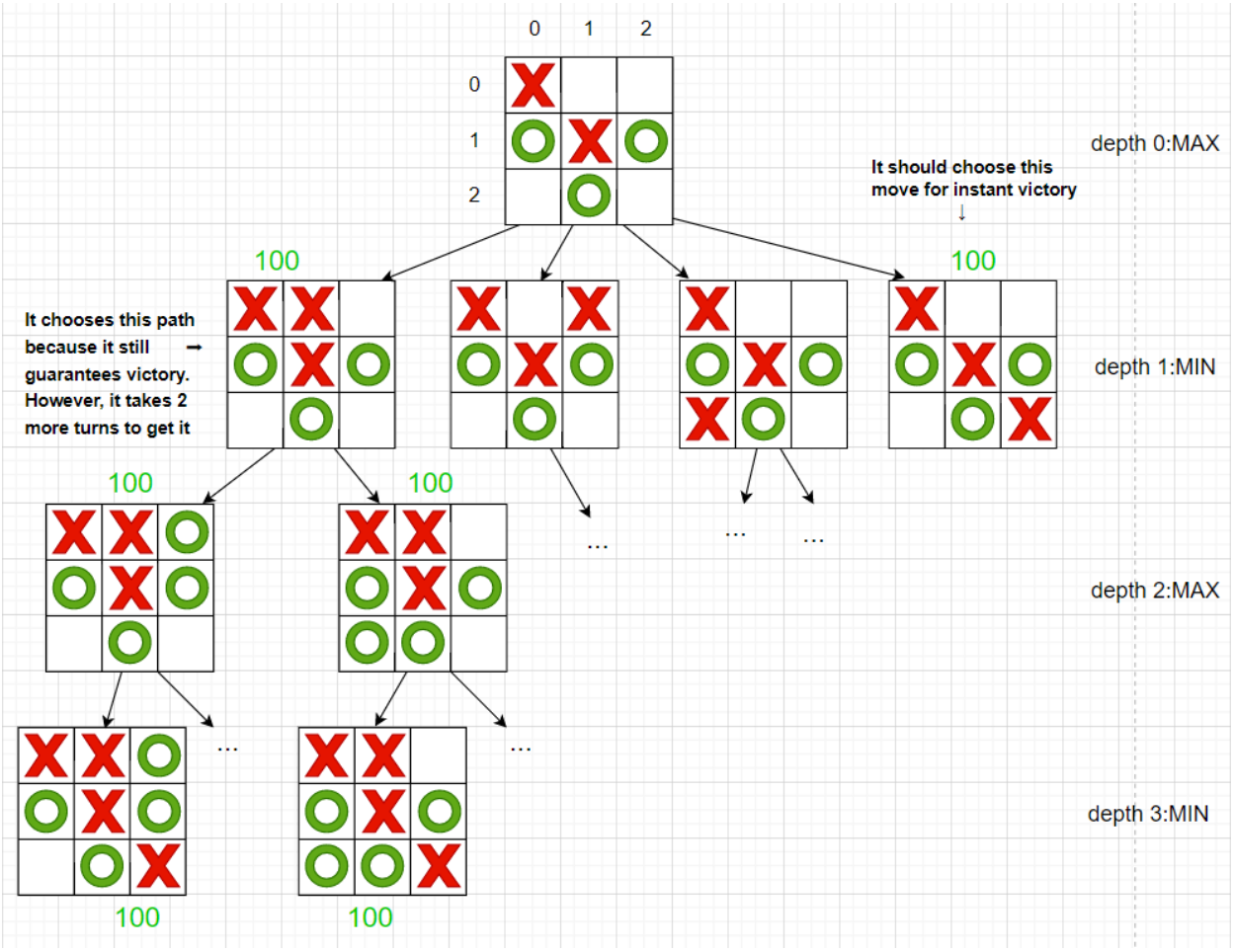
- For example, in the game tic tac toe, suppose the algorithm will find a way to win for player X. It tries to place a move and checks the result, if there is still no result, it will change its role to player O and continue to place a move. When there is a result, a score of 100 represents the victory of player X and otherwise, -100 means that player O wins, and if the result is 0, the game is considered a draw. After having the results, it goes back and calculates the results of the move in the upper node step for step, finally deciding to give the best move which in the case of the image above will be the 3rd move from the left to right (depth 1).

- Below is the basic minimax pseudocode in tin tac toe game :

```
minimax(board_game,depth,MAX):
    result = check_board(board)
    if result != None:
        return result
    if reached depth_limit or no more available move:
        return 0

    if MAX:
        score = -INFINITY
        for move in all_possible_moves:
            value = minimax(board_game,depth+1,False)
            if value > score:
                score = value
        return score
    else:
        score = INFINITY
        for move in all_possible_moves:
            value = minimax(board_game,depth+1,True)
            if value < score:
                score = value
        return score
```

- However, in this project, we need some improvements to the algorithm so it can be 'smarter' and decide the best move. Let's examine the game board case below, when player X is just one step away from winning, Minimax will traverse the moves in the order of each available cell from left to right in each row and consider each row from top to bottom. Therefore, the cell in row 0 column 1 will be selected first because the result it returns is still the best (100 points - guaranteed to win) and skip selecting the cell in row 2 column 2 which will give it an instant win.



- So in order to increase 'the intelligence' of Mini-Max, we have to tell it that it should choose the best and least expensive step through consideration of depth:

```
result = check_board(board)
if result != None:
    if result > 0: return result - depth
    elif: result < 0: return result + depth
```

- In addition, this project adds another element to Mini-Max that the next moves will be shuffled before traversing all these moves. This increases the probability of choosing the best move while also creating new ways to play:

```
for i in random integer number from 0 to 2:
    for j in random integer number from 0 to 2:
        try_this_cell(i,j)
```

- With Mini-Max algorithm, the computer can make the best move decision in game. But the thing it has to consider 9! = 362 880 nodes for a 3x3 board or 25!=15 511 210 043×10^{25} nodes for a 5x5 board is too much and time consuming. So we need some techniques to enhance the performance of Mini-Max.

b. Alpha-Beta Pruning:

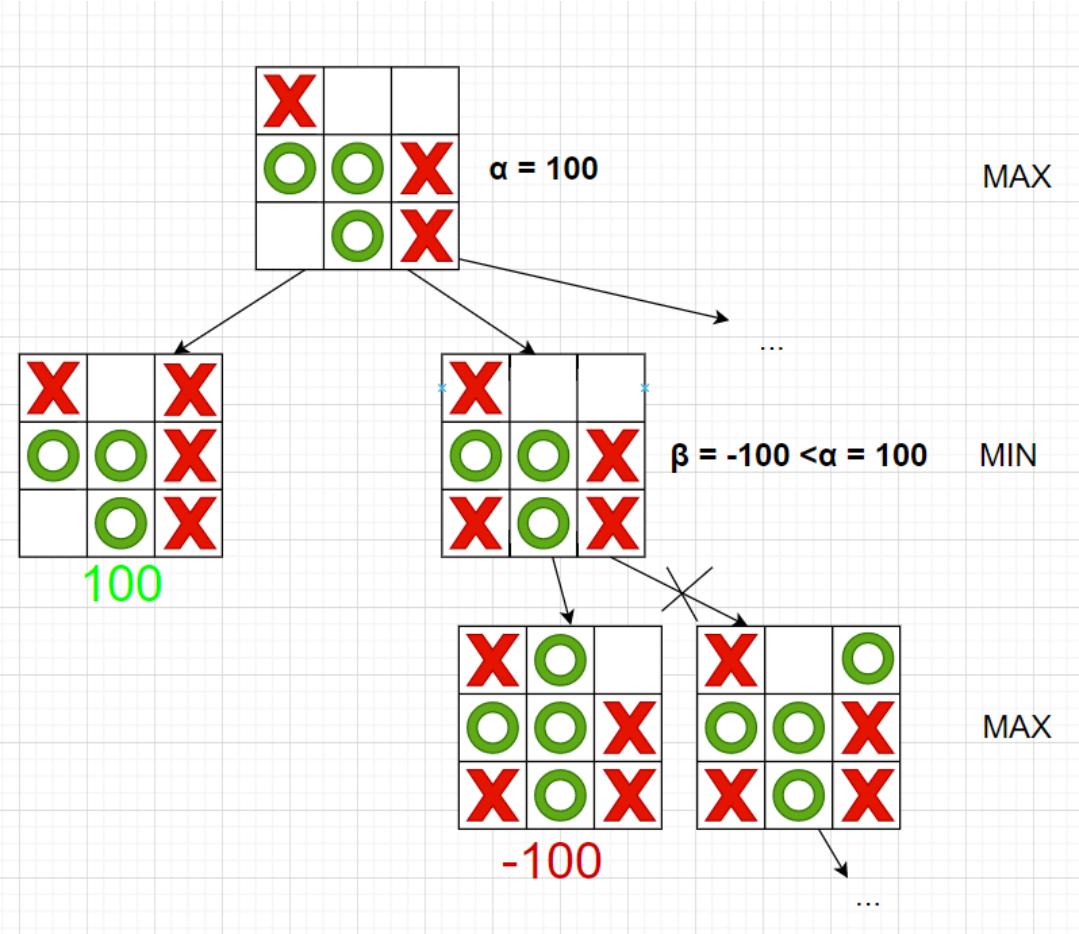
- Alpha-Beta Pruning is a technique used to optimize Mini-Max algorithm. Since Mini-Max's game decision tree is huge and complex with some useless branches, we need to 'pruning' them to avoid checking on these branch. This technique will add 2 more parameters to Mini-Max called ***Alpha*** and ***Beta:***

- **Alpha** is used and updated only by MAX and it represents the maximum value found so far
- **Beta** is used and updated only by MIN and it represents the minimum value found.

The condition for alpha beta pruning to perform pruning useless branches is:

```
if beta <= alpha:
    break
```

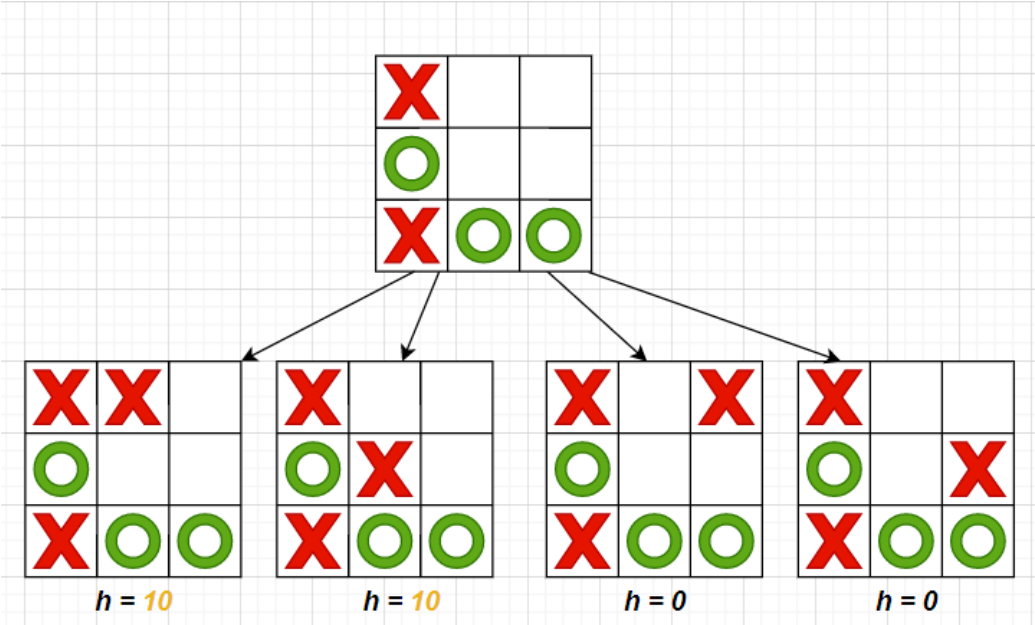
- Follow the example game board below, Mini-Max performs a depth first search with two initial parameters : $\alpha = -\infty$ and $\beta = +\infty$. When it reaches the leftmost leaf node, it returns 100 and is assigned to α ($100 > -\infty$). This means that the largest value currently found by MAX is 100 and it should only consider possible values greater than 100. Go to the next node and dig down that node, MIN finds the value -100 and assigns it to β ($-100 < +\infty$) . When it comes to this, MAX needs to take care of values that can be greater than 100 while MIN can only yield values less than or equal to -100. Therefore, it does not need to consider other sub-paths of MIN.



- Thanks to Alpha Beta Pruning, we have reduced a large number of nodes that have to be explored and we can still find the best result. This technique is already good enough for Mini-Max algorithm to play tic tac toe on 3x3 maps both smartly and quickly. But for a 5x5 map, the performance still hasn't improved significantly. We need to continue to explore more techniques to boost Alpha Beta Pruning and Mini-Max algorithm.

c. Heuristic Improvement:

- Reducing sub-branches with Alpha Beta Pruning has saved considerable time, but what if the value is selected from the beginning is the best and pruning is done continuously from the beginning to the end ? An additional improvement for Alpha Beta Pruning to work best is sorting the next moves based on the heuristic value. This only helps Alpha Beta Pruning cutoffs many times but doesn't affect the accuracy of Mini-Max algorithm.



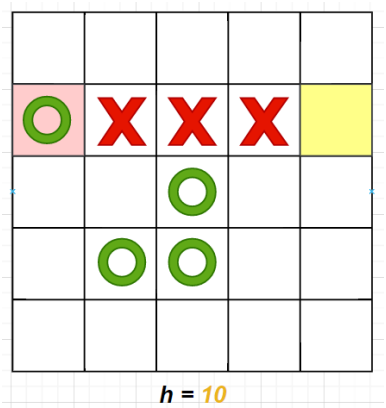
- All four next moves of the current board above still have no value to return. But looking at all of them, there are two moves that can be close to victory, a move that forms 2 marks in horizontal row and a move that forms 2 marks in diagonal row. Moves that are close to the goal will have a heuristic value of 10, the other normal moves have a heuristic value of 0. This ensures that the first move choice is always the best and drastically reduces moves that are useless or lead to loss.

- All we need to do now is building a heuristic function that is admissible. In this project, the heuristic value of 3x3 map will be based on the following simple conditions :

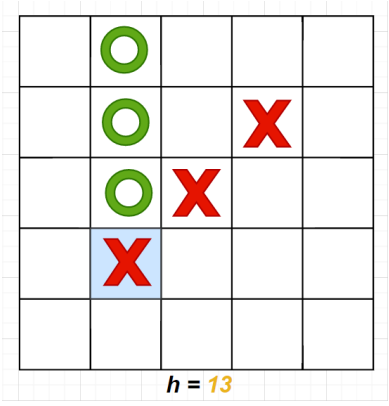
- If this move doesn't bring anything, it will score 0.
- If this move helps MAX form 2 marks in in horizontal, vertical or diagonal row, it will score 10.
- If this move helps MIN form 2 marks in in horizontal, vertical or diagonal row, it will score -10.
- If this move has result, its score is 100 and -100 respectively with win of MAX and MIN.
- If this move at center cell of the board (a cell that can lead to 4 more chances of winning than other cells), its score will be added 1.

For the 5x5 map, to ensure building the best heuristic function as possible, the moves that not only can form a winning path but also be located at 'strategic position'. The heuristic value will be calculated according to the following criteria:

- If this move has result, its score is 100 and -100 respectively with win of MAX and MIN.
- If this move helps MAX form 3 marks in in horizontal, vertical or diagonal row **and** there must be at least one available cell at either end, it will score 10.
- If this move helps MIN form 3 marks in in horizontal, vertical or diagonal row **and** there must be at least one available cell at either end, it will score -10.



- If this move is located in the inner 3x3 square (or is located at row i column j with $1 \leq i \leq 3$ and $1 \leq j \leq 3$), its score will be added 1.
- If this move lies on one of the two main diagonals (or is located at row i column j with $i = j$ or $j = size\ of\ board\ game - 1 - i$), its score will be added 1.
- If this move has the mark of an opponent already lying nearby, its score will be added 1.



- Also in this project, in the 3x3 game board, there will be no limit on depth because the amount of nodes is negligible and ensures it makes the best decision. But in the 5x5 game board, the limit for depth will be set and gradually increase as the game gets closer to the end. The combination of Alpha Beta Pruning and Heuristic Improvement will help the computer make quick choices and become invincible at the 5x5 map tic tac toe.

4. Analysis of Algorithm:

a. Time Complexity:

- We will set *N* to be the size of the board (N can be 3 for 3x3 maps or 5 for 5x5 maps), the number of cells on the board will be *n* ($n = N^2$ in the initial game) and *m* will be the number of marks placed on the game board at a moment. Suppose we take a constant μ time to compute the result of a node. With basic Mini-Max algorithm, in the initial game, the root node of the game tree (at depth 0) will generate *n* nodes (at depth 1) and each of those child nodes will generate the next *n* - 1 child nodes (at depth 2) and so on. When we reach the leaf node, we just need to calculate the result of that node and it takes us μ time to do it. We backtrack and the time to calculate the upper node value based on the child nodes will be $\mu * (n - depth)^{d - depth}$ (with *d* is the maximum depth of the tree). Now we come back gradually to the root node, we can respectively calculate the time at depth 2 is $\mu*(n - 2)^{d-2}$, at depth 1 is $\mu*(n - 1)^{d-1}$ and finally at depth 0 (at root node) is $\mu*(n - 0)^{d}$. Therefore, the time it spends completing the computation of the whole tree is $\mu*(n)^{d}$. The time complexity of the tree with a branching factor of *b* and max depth of *d* is :

$$\mathbf{O(b^d)}$$

- With the help of Heuristic Improvement, the moves will be arranged in order and the first move chosen is always the best, then Alpha Beta Pruninng will reduced the number of leaf node positions evaluated to about O(b×1×b×1×...×b) for odd depth and O(b×1×b×1×...×1) for even depth or:

$$\mathbf{O(b^{\frac{d}{2}})}$$

All the first player's moves need to be traversed to find the best value. For the second player, just find the best value to discard the other moves. Alpha Beta Pruning makes sure not to consider second player’s other moves.

b. Space Complexity:

- At each node of the tree, we need to hold the information of the next nodes at the same depth (sibling nodes) so that after calculating the current node value from value of child nodes, we can move to those nodes. Therefore, with branching factor of *b*, at each depth we have to store *b* and with max depth of *d*, we get the space complexity as :

O(*bm*)

5. Program Demo:

Note : run the program in file **main.py*
Program Demo video: https://youtu.be/N2gdM61TAvw

Image 1 : the main menu of the game

Image 2 : Menu to choose board size

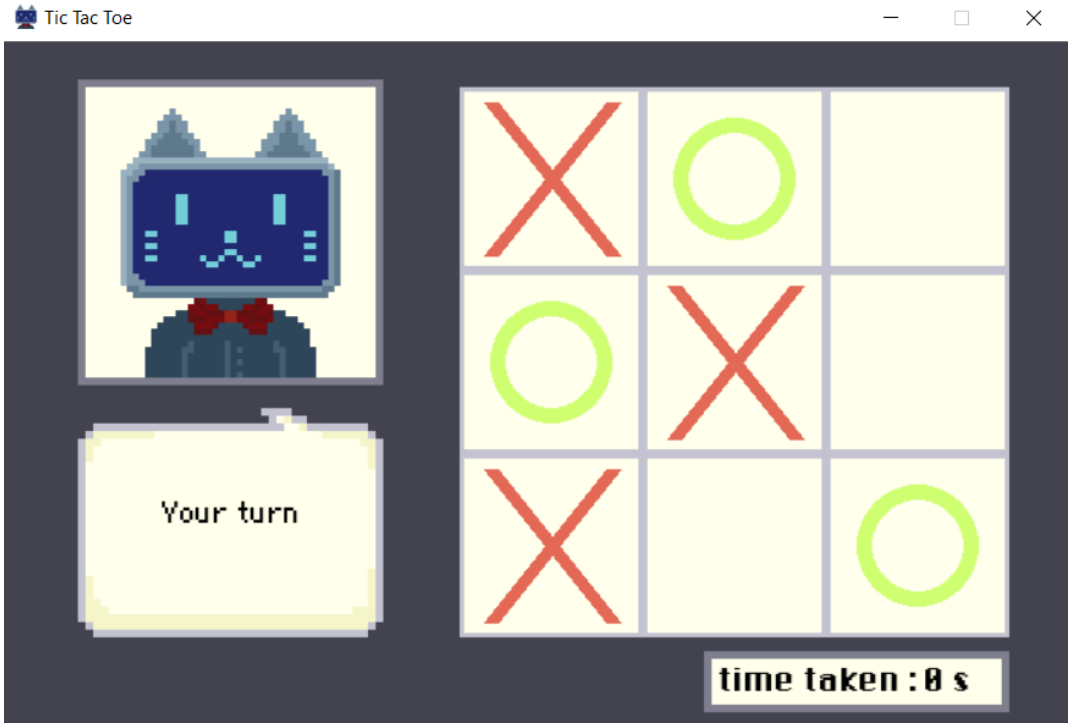


Image 3 : The interface to play tic tac toe in the 3x3 map

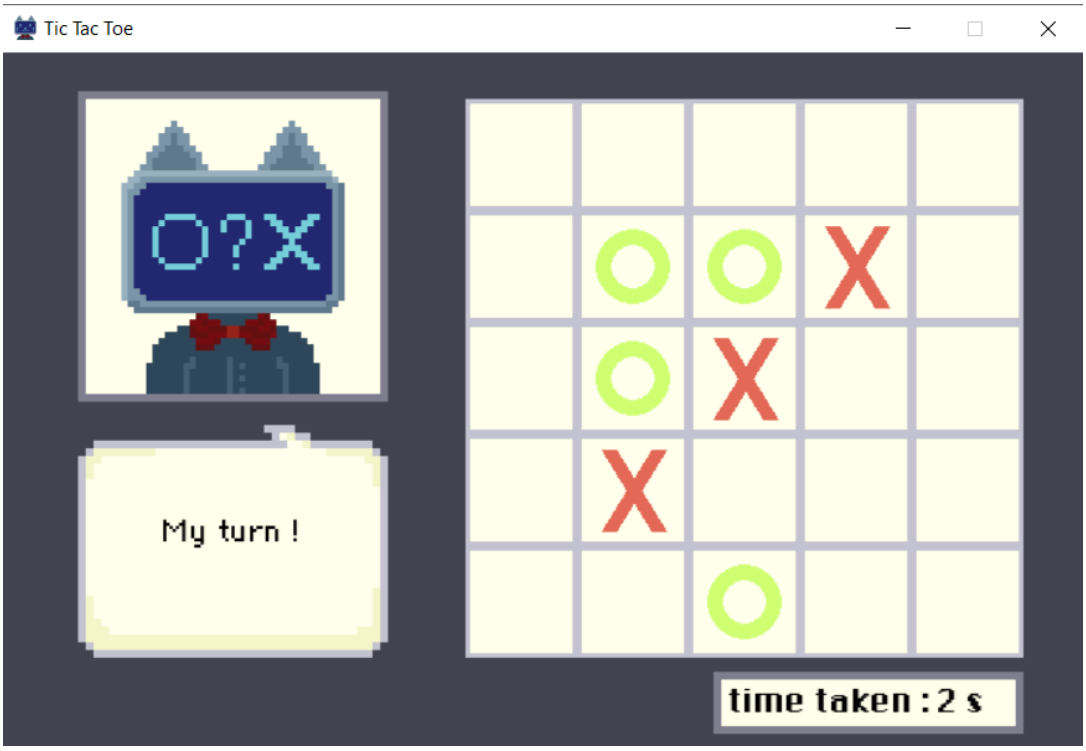


Image 4 : The interface to play tic tac toe in the 5x5 map

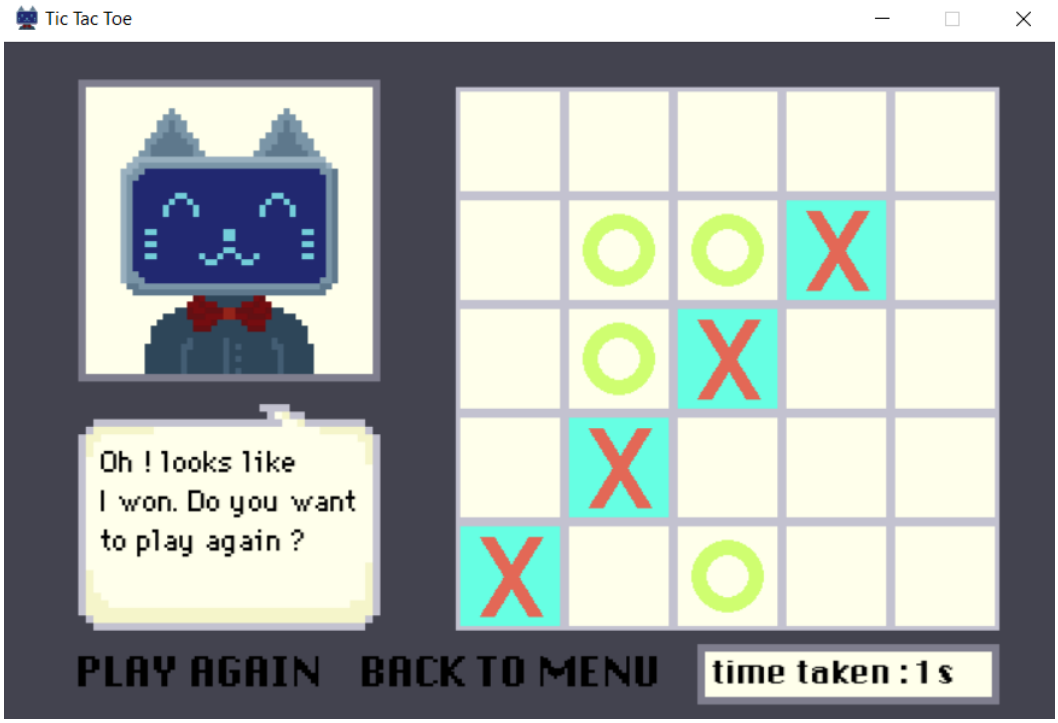


Image 5 : Game over screen with 2 options appear (play again or back to the menu)

6. References:

1. never stop building - Tic Tac Toe: Understanding the Minimax Algorithm – December 13, 2013
2. Lars Wächter - Improving Minimax performance - November 29, 2021
3. Wikipedia - Alpha–beta pruning – 2022
4. Adrian S. Tam - Tic-tac-toe using AI from the last century – March 15, 2019