AI PROJECT REPORT

# **1. Introduction**

## **1.1. "Tic toe toe" game**

Tic-tac-toe is a game for two players, X and O, who take turns marking the spaces in a 3x3 grid. The player who succeeds in placing three of their marks in a line (horizontal, vertical or diagonal) is the winner.

The gameplay of tic-tac-toe is very simple, players will soon discover that the best-play from both parties always leads to a draw. The number of possible games of tic-tac-toe is just 26,830. Due to its simplicity, it is straightforward to develop an AI to play tic-tac-toe perfectly.

This program is mainly an AI for playing tic-tac-toe. Apart from the original game mode, it can be expand into a larger grid.

## **1.2. Problem describing**

Finding the correct move in tic-tac-toe is like child’s play. Because the board is small, one will just end up with one or two options in his/her move without any thinking, and implementing an AI should just be as easy as that.

One of the most important techniques in tic-tac-toe is blocking, it means preventing the opponent from making a line by blocking it with your symbol. The AI should be doing just that, it should detect any marks in a line and stop the opposing player from getting any winning positions.

If there is nothing to block, the AI should know and start building its own line in order to get to a comfortable spot for a win, instead of playing some random moves.

The number of moves ahead the AI can see should be only one, one moves ahead is enough for finding the optimal move in any positions of a tic-tac-toe game. There should be no different if the AI be able to see more than one moves ahead.

# **2. Proposing solutions**

# **3. Algorithms and heuristics functions**

**3.1. Alpha beta pruning**:

**3.1.1. Definition:**

Alpha-Beta pruning is an algorithm that decreases the number of nodes in minimax algorithm in its tree node. It will stop evaluating a move when at least one possibility has been found that the move to be worse than a previously examined move. It also means that the moves need not to be evaluated further.

**3.1.2. Idea of Algorithm:**

Because alpha-beta pruning algorithm has been taught in class, we will not explain about it in detail. The algorithm maintains two values, alpha and beta, which present the minimum score that the maximizing player is assured of and the maximum score that the minimizing player is assured of respectively. In algorithm, initially alpha is negative infinity and beta is positive infinity.

The main benefit of alpha-beta pruning in fact that the branches of the tree node can be eliminated. This way, the search time can be limited. The condition for alpha-beta pruning is when the value of alpha is greater than or equal to the value of beta. Moreover, the max player will only update the value of alpha, whereas the min player will only update the value of beta. While backtracking the tree, the node values will be passed to upper instead of values of alpha and beta. We only pass the value of alpha beta into the child nodes.

There are two cases for alpha-beta pruning:

First, worst ordering that alpha-beta pruning sometimes does not prune any of the leaves of the tree and wastes the time as the minimax algorithm.

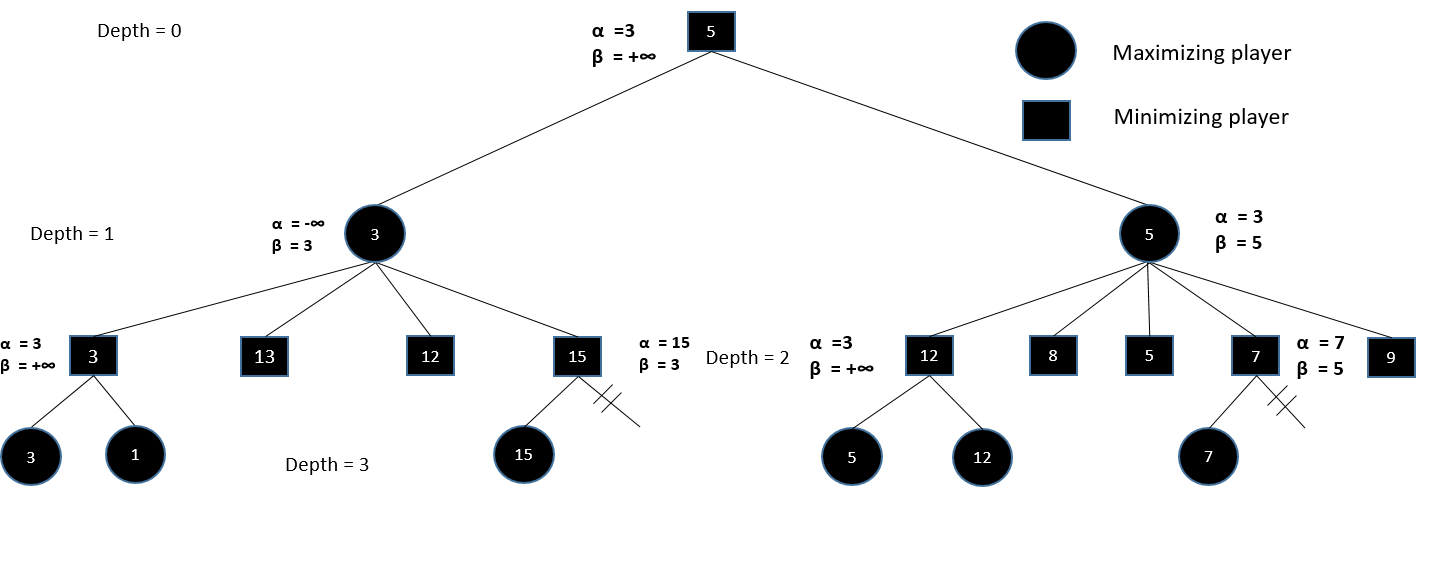
Second, ideal ordering that occurs lots of pruning happens in the tree, and the best case for it is the the pruning happens at the left most node, because we apply the Depth First Search for this algorithm.

**3.1.3. Apply to Tic Tac Toe**

As we has known, Tic Tac Toe includes two player that fill step by step in the Board. Therefore, in term of AI mode, we supposes the O is presented for AI player, and X is for human player. When we apply the alpha beta into the turn of AI player, the O will be the maximizing player and minimizing player for X.

The depth used in our game is 3. We evaluate the state of every available node from the current state by the evaluate function, and from that we apply the alpha-beta pruning to get the best spot for the “first” current state.

The below diagram shows the basic implement of alpha-beta pruning:



As you can see, at the depth = 3 that the value of alpha(15) is greater than the value of beta(3), so the branches to the right of the node(15) is eliminated.

**3.2.** **Heuristics functions**

In the alpha-beta pruning, we use the depth of game is 3, so we need to evaluate of the each board at that depth. The evaluate function called eval() in our code. The evaluate function will assess the value of each board, which based on the following condition:

|  |  |  |  |
| --- | --- | --- | --- |
| Number of X | Number of O | Number of Empty Cell | Point |
| 3 | 0 | 0 | -20000 |
| 0 | 3 | 0 | 20000 |
| 2 | 0 | 0 | -1000 |
| 0 | 2 | 0 | 500 |
| 1 | 0 | 0 | -1 |
| 0 | 1 | 0 | 1 |

The above table show 6 case of each “win direction”, which will explain following table:

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

In our code, we use the function getRightSeq() to generate all the win direction of specific size of Board. At the size of 3x3, we can use 8 ways to form 3 X or 3 O in them; and they are: 123,456,789,147,258,369,159,357. The evaluate function will return the sum of all 8 sequences by counting the number of O and number of X each.

# **4. Implementations**

In our program, we use JavaScript as the main programming language. Besides, we use html and css3 to build the front-end of our game, this brings a familiar and easy-to-use interface for users.

Most game programs, in fact, most interactive programs of any kind, consist of an initialization section followed by a sequence of steps carried out repeatedly. In the case of the tic-tac-toe game, the overall program structure will be something like this:

1. Initialization

2. Get opponent’s moves

3. Compute program’s moves

4. Draw moves on screen

5. Check terminal state and inform the winner.

Figuring out the computer’s strategy is the most complex part of the program’s job. But this strategic task is still only about a third of the complete program. The five groups are quite cleanly distinguishable in this project. There are relatively few procedure invocations between groups, compared to the number within a group.

The following is a brief description of the …. functions needed for the program.

✦ size(choice) : take the option of users, which includes size of board

✦ createBoard() : creates a board of size n \* n.

✦ option(type) : takes the option of users, which includes number of player.

✦ go() : run the game

✦ startGame() : initializes a new game, create board, create combo of win directions corresponding to the size of board.

✦ turnclick(cell) :

The function fill the X or O in clicked cell corresponding to turning of player if the type of game is two player.

The function fill the X first in the clicked cell and then fill O into cell which is returned from bestspot() if the type of game is one player.

Both of them fill the X or O by the function turn().

✦ turn(id, player) : fill X or O in cell by its id.

✦ checkWin(Board1) : This function returns True if there exists at least 1 win-combo.

✦ checkTie(): This function returns True if the game is in the tie-state or time up.

✦ \_checkTie\_(Board) : Same feature with checkTie(), but it is used for alpha beta function. It means that checkTie not declare.

✦ declareWinner(player) : informs the winner.

✦ find\_min(list) : return the min value for AlphaBetaPruning.

✦ find\_max(list) : return the max value for AlphaBetaPruning.

✦ Find\_available\_node(\_Board) : find all available nodes .

✦ bestSpot(\_Board, player) : finds the best spot to click for AI player.

✦ alpha\_beta\_pruning(\_Board,depth,player,alpha,beta):

return the best spot to bestSpot() function.

✦ eval(\_Board) : evaluate the current state of Board and return the bestspot.

✦ Point(\_Board) : return Point of state.

✦ calculate(numX,numO) : calculate the point for function Point.

✦ getRightSeq(length\_win) : This function returns the win combo of a specific sizeofBoard.

✦ horizon(length\_win) : find all the horizon direction of specific sizeofBoard.

✦ vertical() :find all the vertical direction of specific sizeofBoard.

✦ diagonal() :find all the diagonal direction of specific sizeofBoard

And the logic functions:

- start() : start a new game.

- continue\_game() : continue to the game playing.

- restart() : restart the game with blank board.

- backtoMenu() : back to the main menu.

- pause() : pause the playing game.

- resume() : continue game after pausing.

- hoverOver() : makes the board display “faint” X/O corresponding to the current turn.

- hoverOut() : remove the “faint” X/O.

- playSound() : creates sound after clicking.

# **5. User interface.**

First, you need to click Start to start a new game. If you haven’t known

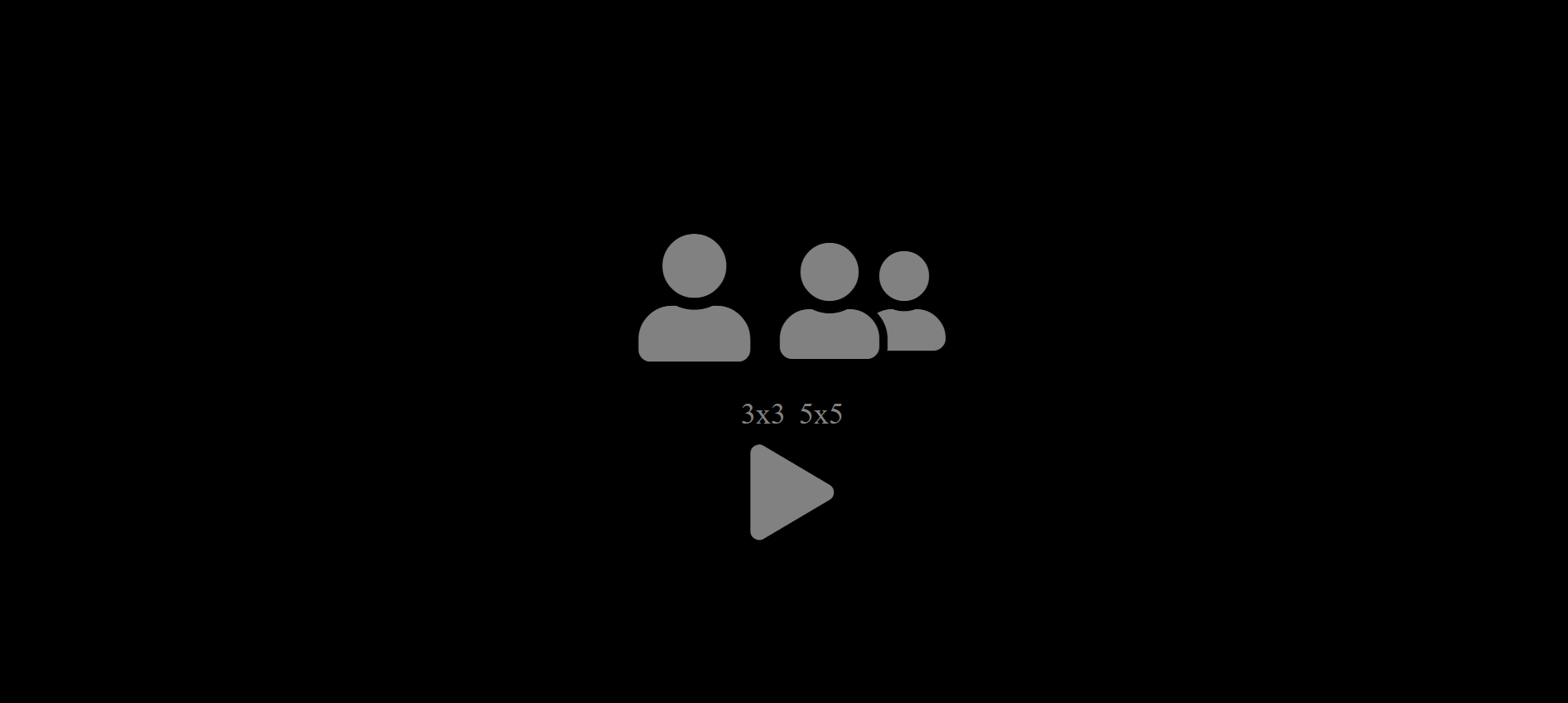
about the rules of Tic Tac Toe game, you can click on “ How to play”.

If you want to know more about our team, you may want to be click on

About in the initial screen.

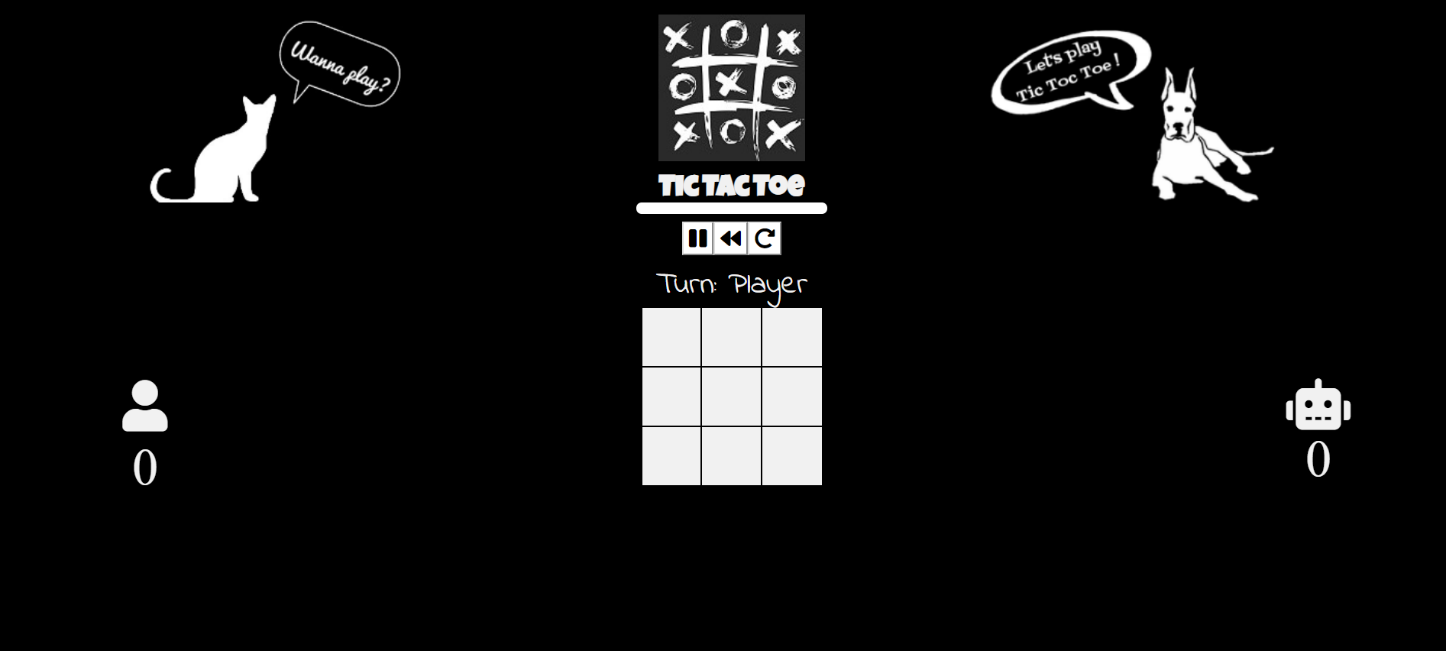


Next, you need to choose the size of board and game mode to start a new game.



Web is used as the game interface for its flexibility and user-friendliness. Two games modes can be selected by the user: Human - Human and Human-AIplayer.

**The main playing screen of Tic Tac Toe:**

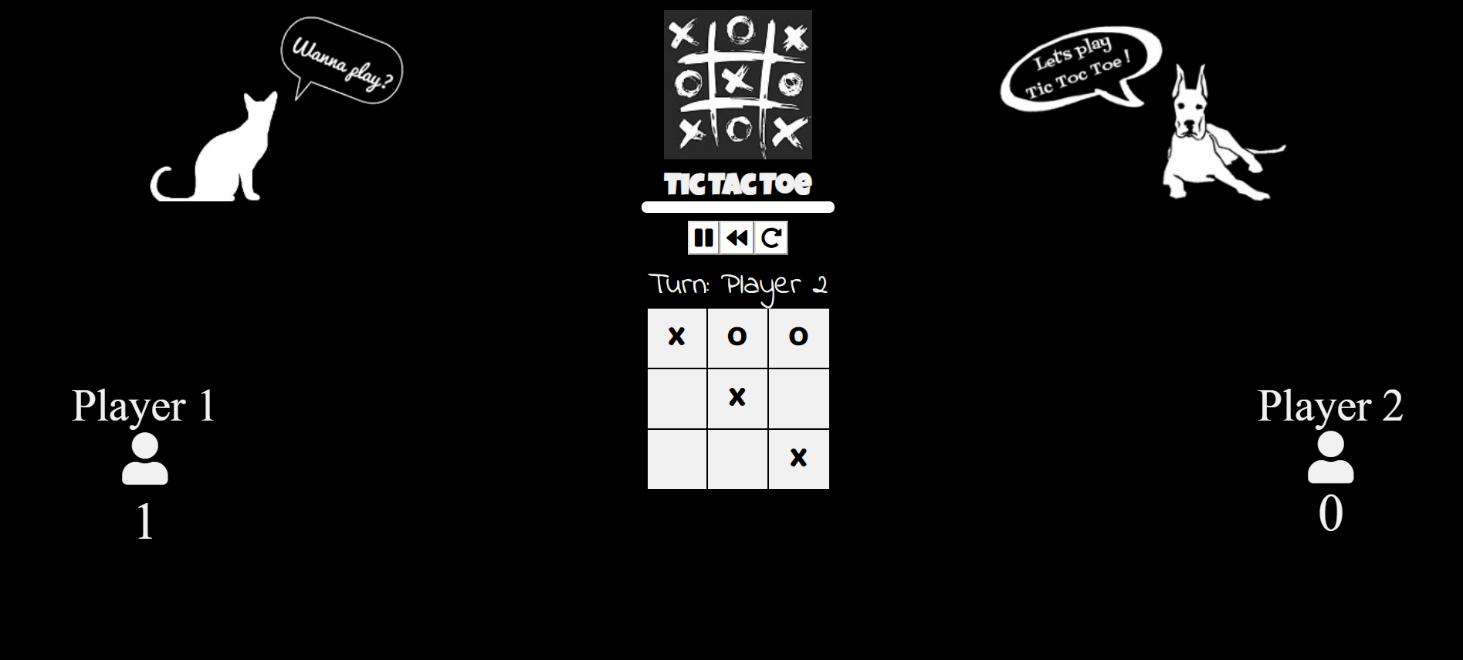
****

**In Human-Human mode**:

2 players take turns playing with each other by their own strategies, after one

person finishes his turn, the other person will take his turn. The game comes to terminal state when either first fills in 3 cells in diagonal, vertical or horizontal direction or the board is full of X/ O but nobody wins.

For example: This happens when the first player wins.

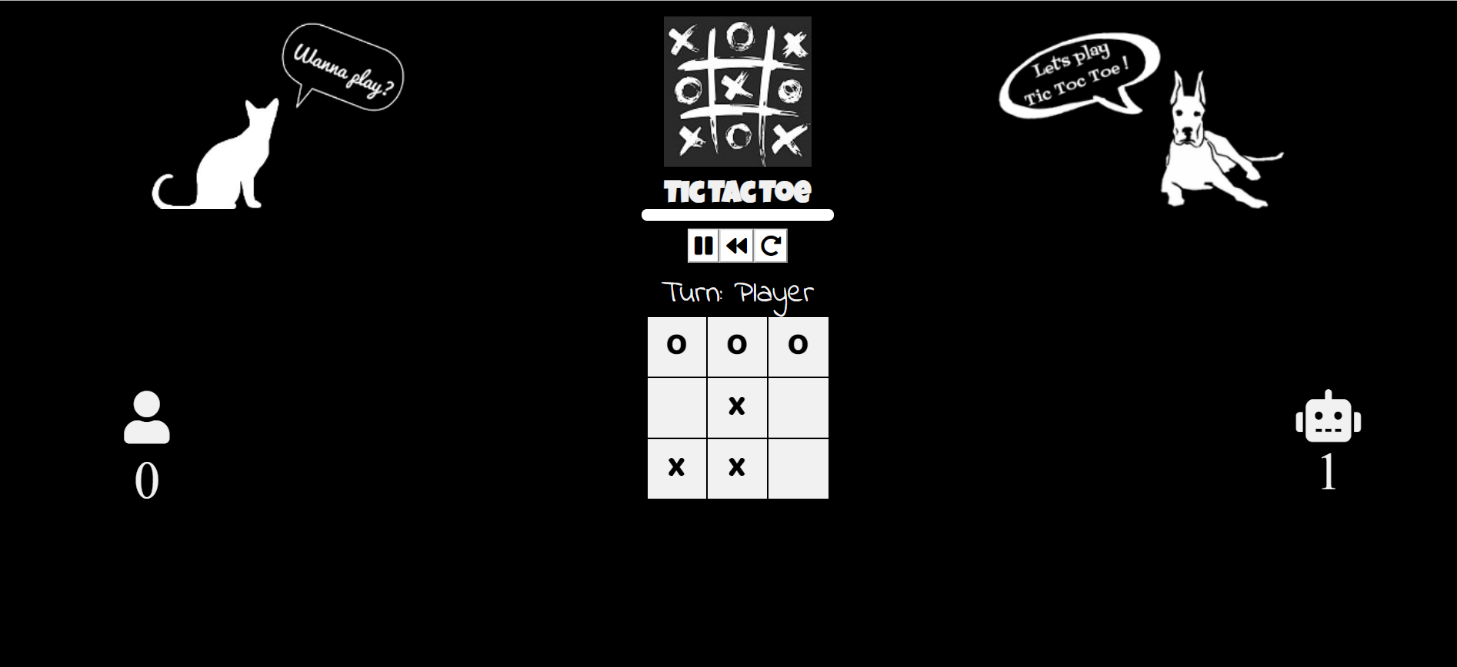


**In Human-AI\_player Mode:**

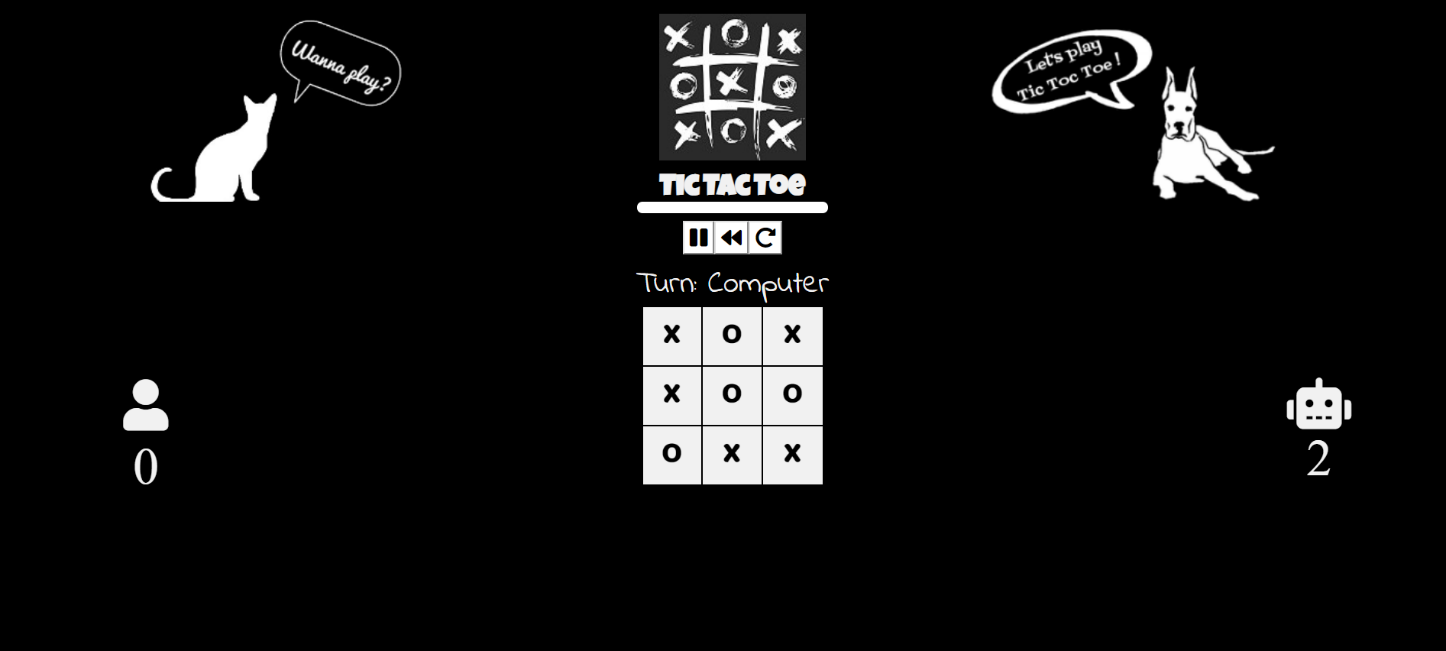
* Human player will take his turn first. AI-player will calculate to find the best spot to fill in. The best spot is found by alpha-beta pruning and minimax algorithm that was performed in section 3.

The task of program is making AI\_player be in the win/ tie mode.

* Winning mode:



* A tie/draw:



Our goal is that the losing state will not appear in the human- AI\_player

mode.

# **6. Result and demos:**

In order for the computer to make "smart" moves as possible, we apply the Alpha - Beta pruning algorithm, with the advantage of easy installation and fast computing speed.

If both the player and the AI play optimally then it is destined that you will never lose (“although the match can still be drawn”). It doesn’t matter whether you play first or second.In another ways – “ Two expert players will always draw ”. However, the AI player will never ignore the chance to win the games, so you need to be careful of each move like an expert.

Demos:

The tic-tac-toe is a basic feature in applying the alpha-beta algorithm in the search for the most optimize solutions. By this algorithm we can also find the best

The demo of the tic-tac-toe will be shown in our presentation in class.

# **7. Evaluate the result:**

After understanding about the algorithm in the tic-tac-toe, we can conclude about its characteristics

|  |  |
| --- | --- |
| Evaluation | |
| Sufficiency ? | Yes |
| Optimization | Yes |
| Time complexity |  |
| Spatial complexity (depth) |  |

About the problem that can we expand the size of Board, we had built the flexible functions. The size of Board and the winning directions will be changed base on your choice.

# **8. Reference:**

PDF about Minimax và implementation of Alpha-beta (in Vietnamese):

<http://staff.agu.edu.vn/nvhoa/AI/lecture4.pdf>

Alpha-beta pruning:

<https://www.javatpoint.com/ai-alpha-beta-pruning>

<https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/>