International Journal of Scientific Research & Engineering Trends
Volume 6, Issue 4, July-Aug-2020, ISSN (Online): 2395-566X

A Game Based Implementation of Minimax Algorithm Using AI Agents

Ishank Lakhmani, Vartika Punjabi
Dept. of Computer Science and Engineering
VIT University, AP, India
ishank.lakhmani@gmail.com, vartikapunjabi1@gmail.com

***Abstract*-This paper reports our analysis of results obtained after applying minimax algorithm for the implementation of an ultimate tic-tac-toe. We use a heuristic function to obtain the best possible move and, as soon as the game ends, then the updated method is propogated from the final move to the first move using backward chaining. That is how the agent learns. The performance of the agent is assessed by taking full-board and partial board representations. The game developed allows agent to play against other AI agents and human players also hum players to play against other human players.**

***Keywords*-artificial intelligence, backward chaining, minimax, ultimate tic-tac-toe**

I.INTRODUCTION

Ultimate tic-tac-toe (UTTT) an extension of the NxN TTT, where there are 9 blocks each having NxN cells. Its a 2 player game. The first player has to play with crosses (x), and so the opponent will play with circles (O). The two players alternate turns and each player places his crosses/circles according to certain rules. The basic intuition of both the rules is that a player placing a circle/cross in a particular cell of a NxN block, determines the BLOCK(s) where the opponent should move next. Basically, a cell in any of the NxN boards, can be thought as a representative of the position of the 9 blocks in the bigger board. Winning the board is similar to that winning a NxN tic-tac-toe, but winning a board is not enough.

To win the game, a player should win the bigger board. Similarly the game board can be considered as a tree where the initial state of the board is the root node. All future possibilities are considered as the child nodes. This further expands to cover all the possibilities for the final state for the game. It thus becomes easier to obtain an optimal move after the construction of the tree is completed.

Since the game consists of NxN different blocks of TTT which are being played in a parallel manner, a player of the game must take into consideration all of the different blocks while choosing his move. Not only that, but from the nature of choosing the block a player must play in, he must also consider the long-term effects of his move. We have implemented 2 bots simultaneously using minimax search algorithm which we compare against simulation of the game played by two random agents. UTTT game fits in the adversarial search section minimax search is the main approach taken for designing the UTTT agent. Our aim is to build an UTTT player which has both high winning rate and also runs in a reasonable amount of time. To achieve this, we need to achieve 2 milestones in the implementation process which are: first, the inner evaluation function uses a guide to search and second, custom optimizations on the searching algorithm. Some of these were learned throughout the period of our course and some are unique ideas to develop our AI agent. In this paper algorithm for the UTTT game is discussed in section 2 followed by a detailed flowchart in section 4. Section 6 highlights the simulation results and the paper is concluded in the following section.

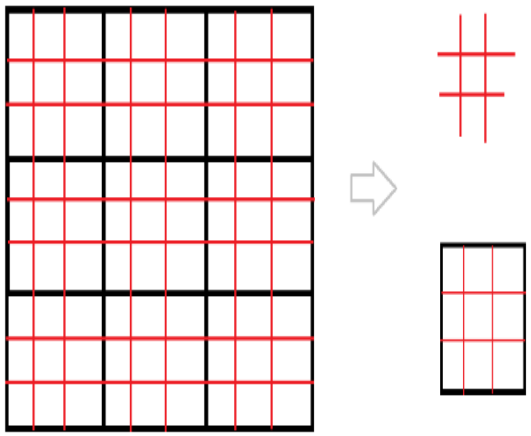

Fig.1 structure of ultimate tic-ac-toe

II. ALGORITHM FOR UTTT

The position of cross/circle placed by one AI agent highly influences the position where the other AI agent can place his cross/circle .figure 2 represents all the positions (blocks) where the player is allowed to play, in brown, after the 1st agent makes its move.

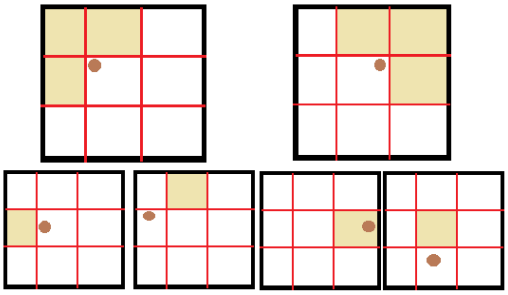

Fig.2 Possible moves

If the player places a cross/circle in any one of the non-corner cells of any block, then the rules showed in figure 2 are followed. however if the player places a cross/circle in any one of the 4 corners of any block, then the opponent is allowed to choose open cells from 3 blocks to make the move.

If there is no place in the given block or the blocks determined by the player, then the opponent can choose to place his cross/circle in any of the empty cells in any block. If the given block is already won, then no moves can be made in that block, even if it has an empty space. The player who wins any 3 blocks of the game which are arranged as a row or column or along a diagonal wins the game. The possible winning block scenarios are shown in figure 3 for agent to win the entire game of UTTT.

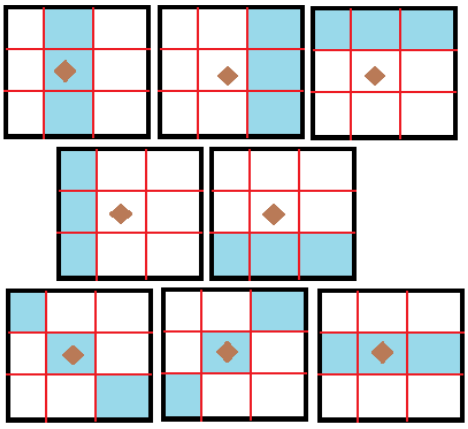

Fig.3 Winning states

Minimax algorithm is based on backtracking approach, used for decision making and game theory, to find an optimal move for a player. The minimax algorithm assigns the two players are a maximize or a minimize. The maximize will try to obtain the maximum score whereas the minimize will obtain the minimum possible score. In every game, there is a specific value associated with each board state. For any given state the score moves towards a positive if the maximizer has upper hand else it moves towards a negative score. The board values are calculated by a unique heuristic function.

III. IMPLEMENTATION OF SIMULATOR

We have implemented a 4x4 UTTT problem in which for every 16 maor cells, there is another 4x4 TTT problem. We intend to implement a bot that can smartly tackle this UTTT problem against another ai-bot or a human player. We use real time heuristics at every step to calculate the next best move to use in order to simultaneously take into consideration a major cell as well as the sub-cells.

The heuristic we use must be weighted and recursive in nature. That is the heuristic for a specific move must also involve that of its sub cells. For every move, the best heuristic score corresponds to the best possible move. The code is written in python. a function first takes the current board scenario, 2D list of NxN elements, and then changes it to a 1D list, which has information if the bloc is already won.A tuple is created which holds the last position played by the agent. This is then used to evaluate the destined block for the next move, and then return an empty location.

A time limit is set for each agent, of 6 seconds. If the time taken to receive the move exceeds the given limit then the match will be exited and the opponent will win the game. On winning the game 4 points are received and the opponent receives nothing. if there is a draw then the team with more blocks will be given 2 points and the opponent given 1 point. if still a tie occurs then the team with more corner cells is given 2 points and opponent receives 1 point. else both teams receive 1 point.

IV. FLOWCHART

The algorithm for the Ultimate Tic-Tac-Toe game is presented using a flowchart in figure 4. Agent 1 or Agent 2 makes the first move. After granting the first move to one of the agents the board is analyzed to check if the winning position is created. If a winning situation is created by any of the agents, then the game terminates else the algorithm checks for two crosses or noughts. There exists a case, after the 4th move, where winning chances are reduced. Thus it is required to evaluate the steps which will help the agent 1 to win, so the board is checked for winning or similar situations. If the next move does not generate a winning situation, then block 6 evaluates the further lead to win.

If there exists no such move then block 8 is used to evaluate the move for creating win situation for the opponent. If further no such situation exists then a simple move is made, creating two in a row in block 10. This is considered as creating a win like situation. If now a winning like situation is observed for the opponent, in block 8, then the winning position is forked in block 9. Then block 6 is checked again for detecting a win like situation that is 2 blocks in a column or a row or diagonally across the board, and a third position is vacant

then that position is placed in block 7 transferring further control to block 5, which checks for a win.

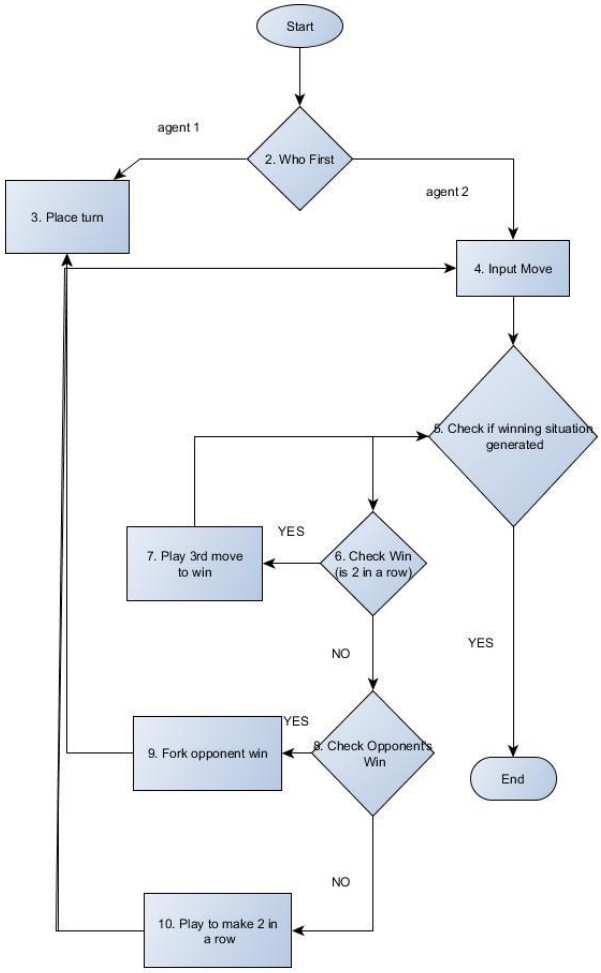

Fig.4 Flowchart.

V. EVALUATION FUNCTIONS

For assessing our UTTT board we have used three evaluation functions. They are as follows.

1. Cells Weight Heuristic:
This strategy considers the closest move to estimate an optimum next move i.e. it looks at the board and finds the most desirable state of the board based on its previous move. we can conclude the following from this implementation.1. The board gets the highest score possible if the last played move was the winning move. 2. State gets a lesser value, still a high one, if the last move will block the opponents move for an immediate win. Immediate win is when the opponent had cells of a row or column and the move to be played would have been in the 3rd cell. 3. If none of the conditions is true then the weight is assigned according the weight function, which will evaluate the desirability of the any cell, and then pick the cell with the highest weight.

2. Recursive Weight heuristic
This is a complex function which involves assuming a winner. The winner is assigned a positive score and negative score for the opponent. If the board is decided as a tie we pick a neutral score. We recursively call a heuristic value, calculated using weight of the board and the values of each mini board. The weight of the board is calculated by taking the sum of each cells of the mini board which is owned by the player and subtracting it with the weight of the cells owned by the opponent. This function is inclined towards calculating winning but also calculates an evaluation for the other cases.

3. Winning Possibilities Heuristic
Like the previous heuristic function, recursive weight heuristic, it performs calculations in a similar fashion. The player gets the maximum score if it is won by the player else a very low score, and zero if the game ends in a tie. First the value for the UTTT board is evaluated, by considering it a TTT board itself, by using the weight and value of each mini board. The value of mini board is calculated by estimating the winning possibilities. We assume that the winning possibilities are also occupied the opponent and thus preventing a high estimation.

VI. SIMULATION AND RESULTS

Our project is a menu driven simulations, which asks the user for the type of game they want to play. First we design 2 different types of agents:Random Agent - which chooses random action from the list of legal action. Two regular Minimax (Alpha-Beta) agents with depth-limit of 2. Each agent uses the algorithm discussed in Chapter II and learns new improved moves using backward chaining. The two simulator outputs are discussed below. First case involves when there both are Random Agents and in second case, both are Minimax agents.

Case 1: Random vs Random agent

In this case, two agents(not AI agents) play randomly using random() function and take any random path as a move in their turn. Both players do not use heuristic methods, but play with random turns among them. The algorithm further calculates the min profit for the 'X' player and obtain he position for placing 'O'. Similarly multiple rounds are conducted till a winning state or a draw state is obtained.

Case 2: Minimax agent vs minimax agent

In this case, the game goes on between two agents who have learnt how to play this game, from previous trials and play among them and both agents try to win with the best move possible they can execute. If there is no winning step for the player, the player then makes a move such that blocks the other agents winning move. So, it places a 'O' at a position such that it blocks the computer

from winning. The computer computes the next optimum move and places a 'X'. And the game goes on till a winner is declared.

VII. CONCLUSION

The paper represents an algorithm to train AI agents to execute a game of Ultimate tic-tac-toe game, to completion. The proposed algorithm is implemented in Python using minimax algorithm. Game theory combined with graph theory is used to implement this game. In an ideal scenario the agent evaluates all the possibilities to ensure success whereas in the implemented minimax algorithm it evaluates only one step, to evaluate best immediate move. Even though Ultimate Tic-tac-toe is a small game it is also highly complex to evaluate, hence an unbeatable algorithm can be developed because the state space tree generated will be small. We have used Recursive Weight Heuristics mentioned in Chapter V to calculate the best move to be made by the agent. The algorithm also has a future scope for implementing complicated games like chess and checkers. However as the number of squares on the boards increase so does the number of possibilities.

REFRENCES

[1] Dwi H. Widyantoro&Yus G. Vembrina, "learning to play tic-tac-toe", international conference on electrical engineering and informatics, 2009
[2] Roopali Garg, Deva Prasad Nayak,"Game of tic-tac-toe: simulation using min-max algorithm", international journal of advanced research in computer science
[3] PlamenkaBorovska, Milena Lazarova, "Efficiency of parallel Minimax Algorithm for Game Tree Search:, international conference on computer systems and technologies-compsystech'07
[4] Sunil Karamchandani, Parth Gandhi, Omkar Panwar,Shruti Pawaskar,"A simple Algorithm for designing an artificial intelligence based tic tac toe game", International conference on pervasive computing,2015
[5] Sivaraman Sriram, Rajkumar Vijayarangan, SaaisreeRaghuraman, Xiaobu Yuan, "implementing a no loss state of tic-tac-toe using customized decision tree algorithm",international conference in information and automation,2009.

APPENDIX

The python libraries used are as follows:

- **Python time:** This modules consists of various time-related functions.
- **Python copy:** Assignment statements in Python do not copy objects, they create bindings between a target and an object. For collections that are mutable or contain mutable items, a copy is sometimes needed so one can change one copy without changing the other. This module provides generic shallow and deep copy operations.
- **Python signal:** this module provides mechanisms to use signal handlers in Python.
- **Python random:** This module implements pseudo-random number generators for various distributions. For integers, uniform selection from a range. For sequences, uniform selection of a random element, a function to generate a random permutation of a list in-place, and a function for random sampling without replacement. On the real line, there are functions to compute uniform, normal (Gaussian), lognormal, negative exponential, gamma, and beta distributions. For generating distributions of angles, the von Mises distribution is available. Almost all module functions depend on the basic function random()which generates a random float uniformly in the semi-open range [0.0, 1.0].
- **Python sys:** This module provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter. It is always available.