INTERNSHIP PROJECT DOCUMENT III

ON

AI agent that learns to play a game of tic-tac-toe using reinforcement learning algorithms and improves overtime

Submitted by

Swarupa Balaji

(Artificial Intelligence Intern)

Submitted to:

Founder - Kanduri Abhinay

CTO - Saniya Begam

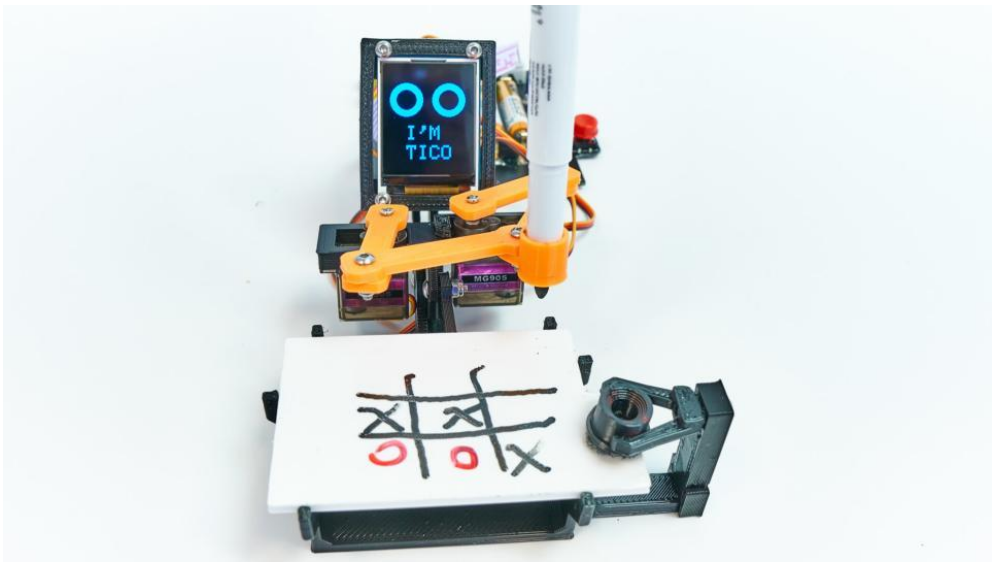
Team Lead – Surya Poojavi

PROJECT III

Title: Tic-Tac-Toe using MCTS

Project Description:

Machines can be trained to learn many things in our everyday life. Some can be trained with or without labels on inputs while some can be trained and improved overtime. Such training algorithms are called reinforcement learning algorithms. I have used Monte Carlo Tree Search algorithm to create a game of tic-tac-toe where MCTS is used to make the model well equipped with all the possible moves in the 3x3 grid. This game is single player, hence the user can play with the AI. The agent learns from its environment and improves its chances of winning overtime.



Software and Tools Used in Project:

Python: Widely used for artificial intelligence projects due to its simplicity as well as extensive libraries can be installed very easily.

Streamlit: A type of open-sourced framework that can be used to create website applications GUI, helps design attractive styles and provides modern templates that can be used to develop model applications.

Monte Carlo Tree Search: MCTS is a type of reinforcement learning algorithm that uses a tree representation of all the recurring moves after a move is played in a deterministic game. Since

game playing has too many possible ways to move around, this algorithm is apt for tic-tac-toe implementation.

Packages and Libraries Installed:

Pandas: For data manipulation and analysis, allowing easy handling of datasets in tabular form. Some techniques include renaming columns, handling missing data, converting data formats and dtypes, loading data, grouping categories etc. Here, pandas was used for creation of the 3x3 tic-tac-toe board using dataframes.

Streamlit: This web development framework was used to display the game implementation as an interactive game for the audience. Streamlit is easy to use and understand, and with a few lines of code I was able to create a game of tic-tac-toe with the user interface.

NumPy: NumPy was used for numerical operations that were used in the implementation of monte carlo tree search. The formula for MCTS uses complex logarithmic functions and numpy helps to simplify them.

Random: The random library is imported to choose any random choices from the best choice of moves in the tree search. It is also used to select the move that would be taken into consideration for applying the rollout simulation.

Math: This library provides a wide range of mathematical functions and constants for arithmetic and more advanced mathematical operations. It is particularly useful when working with mathematical expressions, computations, and algorithms that require precision and efficiency.

Algorithm used in Model:

Monte Carlo Tree Search (MCTS) is a heuristic search algorithm used to make decisions in situations where the outcome of actions is uncertain, such as in games like Go, chess, and video games, or in complex optimization problems. MCTS is particularly useful when there is a vast number of possible moves or choices, and it helps identify the best strategy by simulating future states.

Key Steps in MCTS:

1. **Selection**: Starting from the root node (representing the current game state), the algorithm selects a path down the tree according to a strategy, usually the Upper Confidence Bound (UCB) formula, balancing exploration of less-visited nodes and exploitation of known good moves.

2. **Expansion**: Once a leaf node (a node with no children) is reached, the algorithm expands it by adding one or more child nodes representing possible future states after taking various actions.
3. **Simulation (Rollout)**: A random simulation is run from the new node, playing out a sequence of moves until a terminal state is reached (such as winning or losing in a game). The result of this random simulation (win, loss, or draw) is recorded.
4. **Backpropagation**: The outcome of the simulation is then propagated back up the tree, updating each node along the path to reflect the result. This helps inform future decisions by adjusting the value of the nodes based on whether they led to good outcomes.

Project Execution:

The project code includes two python files, one for the logic and the other for the design.

Logic Code: The logic code has several functions to search, select, expand, roll out and back propagate each state in the tic-tac-tow board.

The formula used for the logic code is based on the **Upper Confidence Bound for Trees (UCT)**. This formula is key to balancing **exploration** (trying out less-explored moves) and **exploitation** (favoring moves that have led to good outcomes in the past). It helps the AI agent decide which node (representing a game state) to explore next.

$$\boldsymbol{Move\ Score} = \frac{\boldsymbol{child\_node.visits}}{\boldsymbol{child\_node.score}} + \boldsymbol{c} \times \sqrt{\frac{\boldsymbol{log(node.visits)}}{\boldsymbol{child\_node.visits}}}$$

The visits and scores keep updating as different moves are traversed one by one and backpropagated to the best serious of moves possible.

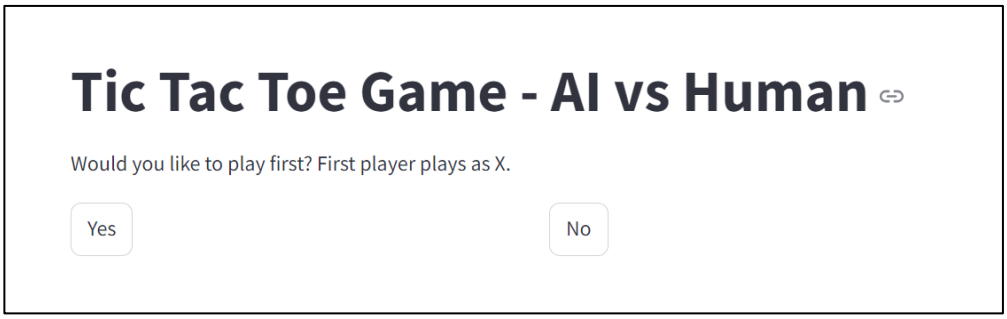
Design Code (UI): The design part uses streamlit as the GUI to present the game in an application with local host. The usual rules of tic-tac-toe are also registered here for the game to comply with them like when it's a draw or win. The core components are:

- **Page Layout**: The app initially asks the user whether they want to play first or let the AI go first. This decision sets up the game state. The board is displayed as a 3x3 grid using an HTML table rendered via Pandas and Streamlit's markdown capabilities. The game board is updated dynamically based on user and AI moves.

- **Game State Management**: The session state is used to store the current game board, the players' symbols, and the current player's turn. This allows the game to maintain continuity across user inputs. The AI uses MCTS to make its move.

- **Game End Conditions**: The app checks after every move if the game has ended, either through a win or a draw. If the game ends, the user is prompted to start a new game with the "Play Again" button.

As MCTS is all about learn as you go technique, different iterations of training the model judges the model performance overtime. With lesser iterations, the model performance was average but as the iterations increased, the model learnt new techniques to defeat the player.



AI Playing out its turn:

A draw state:

Sources:

- Geeksforgeeks
- ChatGPT
- Youtube
- Medium
- Blackbox.ai