My project is an AI for a multiplayer game of Ultimate Tic tac toe.

I am using the monte carlo tree search algorithm since it does not need a complex heuristic function, which would be difficult to make with Ultimate Tic tac toe since its very complex.

I have almost finished my monte carlo tree search algorithm with a normal tic tac toe grid, I will then use this to make the ultimate tic tac toe ai, which would be mostly similar.

There are a 2 main data stores in the ai, which are the game tree, and the node class, which stores each node instance’s parents and children and other values.

There is a get\_child function which uses all possible legal moves to generate all possible children of a node, and adds it to that node’s children, and to the right depth on the game tree.

This get\_child function is used on the root node so the game tree has 1 depth worth of data.

The monte carlo tree search has 4 phases.

The first phase is the selection phase.

Here, a UCT value is calculated for each child of the root node. And then the node with the highest UCT value is returned. Then this function is repeated with the children of the node returned, until the node returned has no children.

So this returns the leaf node with the highest UCT value-path.

The UCT value is what balances searching through paths that are more likely to be good, and paths that haven’t been explored much.

If the leaf node hasn’t been visited before, the simulation phase occurs.

Here random moves are chosen until an end state is reached. Then the value of the end state is added to the leaf node and all its parent nodes, affecting all their UCT values, this is called the back propogation phase.

If the leaf node has been visited before, its expanded by selecting a random legal child node from its possible children and adding it to the game tree and the children of the node.

Then this new leaf node is simulated and back propagated.

This repetition of selection, simulation, expansion and back propogation means, most of the possible nodes can be quickly searched, and the most promising nodes are always chosen for further exploration.

Then theres a make\_move function that looks through the first depth of the game tree for the node that has been visited the most, and this node is used to make the AI move. That node is best because it has most frequently had the highest UCT value so it’s the most promising.

The improvements Im planning on making in the next few days or week, is including some basic heuristics in simulation so nodes aren’t selected completely randomly. And also in the make move function to make sure that a node that 2 in a rows for the opponent are always blocked and so wins aren’t missed.