**Final Project Report**

**Game Mechanism:**

For this project, we are designing a expandable tic tac toe game which is a variant of the original tic tac toe. The two players are allowed to not only occupy slots in the matrix with O/X, they can also place another 3\*3 board next to the original board during each turn. At each turn, a player can have 3 options,  
1. Choose to occupy a slot in current matrix

1. Choose to add another 3\*3 matrix board that shares an edge (left, right, top, bottom).
2. Choose to add a board diagonally (top-left, top-right, bottom-left and bottom-right). But in this situation, for example when you want to add board at top left, the left board and top board are also added if they haven’t added before, therefore you add 3 board in 1 turn. If one of the left and top board already added before, only those “unadded” 3\*3 boards will be added. Which means if you want to add a board diagonally, you will add/complete all 3 boards in that direction and forms a 6 \* 6 board automatically.

There are maximum 9 boards allowed, it forms a 9 \* 9 grid boundary. If a player adds a new matrix, they will lose the rest of the turn.

When the game starts, the program randomly decides who goes first(human player or MCTS).

**Wining Condition:**

The first player who have 3 in a row.

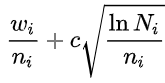
**Draw Condition:**

If the current available board is full(all slots occupied and no one extend any board), or no further extension is possible and no three in a row exists.

**How the MCTS(Monte Carlo Tree Search) works in this game:**  
 Monte Carlo Tree Search is a powerful algorithm for decision making when the game tree is too large for searching. MCTS builds a search tree incrementally and balance between exploration of new moves with exploitation of already explored good strategies. The MCTS contains majorly 4 steps which are Selection, Expansion, Simulation and Back-propagation.

**Selection:**

MCTS uses a function called ucb1 to calculate the score/weight of each exploration in the tree. By definition, w\_i stands for number of wins after exploring node i in the game tree, n\_i means number of simulations/playouts node i has been visited. N\_i is the total visit of the parent node of i.



There is a function our code to always encourage MCTS to explore unvisited route. If the node has not been visited, this will assign it huge value to prioritize it to be explored next.

**if** (node.playouts() == 0) **return** Double.***MAX\_VALUE***;

**Expansion:**

If the game is not end, it will generate all valid child state. In the case of our expandable tic tac toe game, the root of the tree is the original empty matrix board, after expansion, there will be total 17 child node (9 different ways to occupy a slot + 8 different ways to add a board).

We also did an optimization to reduce the redundant process here (normalize()), since the board is 3\*3 and there are lots of symmetric states, the board can be rotated and still be the same. For example, top-left move can be the mirror of bottom right move. Therefore we use a hashset to store identical states that already been expanded. The reason we choose hashset is its fast lookup(O(1) constant) and automatically ignores duplicated values.

**Rollout Simulation:**

Random playout with limited depth based on the expandable nodes. In our game, For each child node in those 17 nodes, we explore the possibilities by setting a depth of 5. For example, if a human player occupy the empty board(root) with X at (row 0, col 0), the MCTS will randomly simulate 5 steps deep, MCTS occupy (0 , 1) -> player occupy (0, 2) -> MCTS:(1 , 0) -> Player:(1 , 1) -> MCTS:(1 , 2). But before we do random simulate, we know that the more simulation MCTS have, the more accuracy(best move) it can get, and there can be still some mistakes since we are doing random moves leading to lots of noises. Therefore we did some optimization for the MCTS in decision making. There are 3 steps ahead of doing random exploration:

1. Check if current player can win immediately or not
2. Check if human will immediately win after MCTS play this move, if yes, MCTS will avoid this option.
3. Thread Detection: check if opponent already have 2 in a row or not(will win soon), if yes, do a block.

After the exploration, we evaluate and reward the algorithm to make simulations more meaningful.

We designed a custom dynamic reward mechanism (evaluate()) based on who goes first since the first player always have advantages.

* We know in a tic tac toe, occupying the center slot gains lots of advantages, therefore we award 10 scores if occupied.
* For those corner slots, we award 5 points each.
* If 3 in a row occurs(someone wins) and MCTS wins:
  + Human goes first: award 100 points.
  + MCTS goes first: award 140 points
* If human wins:
  + Human goes first: penalize 140 points
  + MCTS goes first: penalize 100 points
* We also detect situations when 2 in a row appears(someone about to win soon), and awards differently based on who goes first:
  + MCTS have 2 in a row:
    - Human first: award 20 points
    - MCTS first: award 40 points
  + Human have 2 in a row:
    - Human first: penalize 40 points
    - MCTS first: penalize 20 points

The reason why we set award values like this is because we want MCTS to be more aggressive when it goes first, try to focus on achieve 3 in a row. When human goes first, we want MCTS to be more defensive, focusing on block human’s 2 in a row situation.

**BackPropagation:**

When simulation ends, the result will be propagated back all the way to the root. Each child node’s playout and score will be updated for further selections.

**Time of Runs:**

For time of original tic tac toe game:

We played 1 game with 5 turns until MCTS wins and the simulation of MCTS is set to 50000 times. In each round, the decision time of MCTS is:

1. 3174ms
2. 267 ms
3. 221 ms
4. 222 ms
5. 240 ms

It seems the time for the first step takes significant longer than others and the time decreases after. The reason of this is how MCTS works. Since during the first expansion, there are more available child nodes possible and MCTS will encouraged to explore all the possible routes first due to ucb1 function.

For time of our expandable tic tac toe game:

We played 1 game with 6 turns until MCTS wins, the number of simulation was set to 10000 due to more complex game tree. Each decision time of MCTS are:

1. 6235ms
2. 5619 ms
3. 6038 ms
4. 6282 ms
5. 155 ms
6. 100 ms

This same situation happens, first several decision takes quite a lot of time and suddenly drops.

**Summary:**

In this project, we successfully designed and implemented a expandable tic tac toe game and use Monte Carlo Tree Search algorithm as one of the players against human players. The game follows the 4 standard phases: Selection -> Expansion -> Simulation -> Backpropagation which combines together is a standard MCTS. We introduced dynamic evaluation reward system based on who plays first, adjust the MCTS to be either aggressive or defensive.

The result of the stop watch timing reveals that early move will always takes more time and computations which is exactly the mechanism of MCTS. The result is not perfect since we do randomly move during each simulation, I believe we can implement a smart move strategy in the future which leads to more meaningful and accurate result selection.

**References**

1. <https://en.wikipedia.org/wiki/Monte_Carlo_tree_search>
2. [Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time Analysis of the Multiarmed Bandit Problem.](https://link.springer.com/article/10.1023/A:1013689704352" \l "citeas)
3. [Browne, C. B., et al. (2012). A Survey of Monte Carlo Tree Search Methods.](https://arxiv.org/abs/2103.04931)