# Implement Monte-Carlo Search Tree in Tic-tac-toe

Yuze Liu
Department of Mechanical Engineering
Florida State University

# Game Tree Search

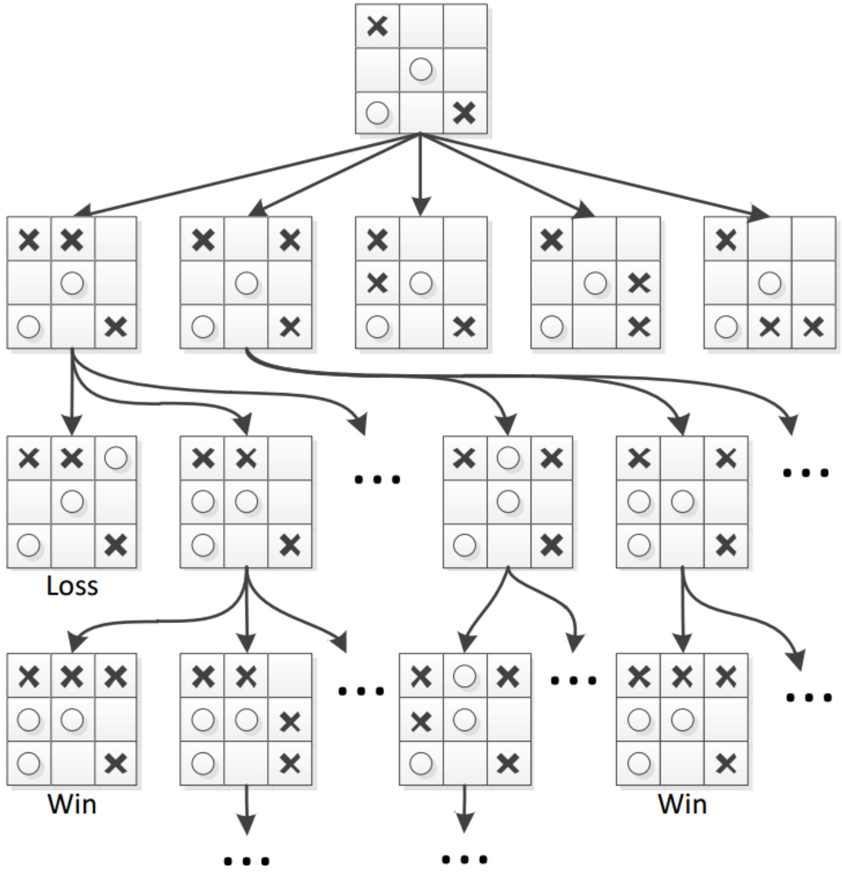


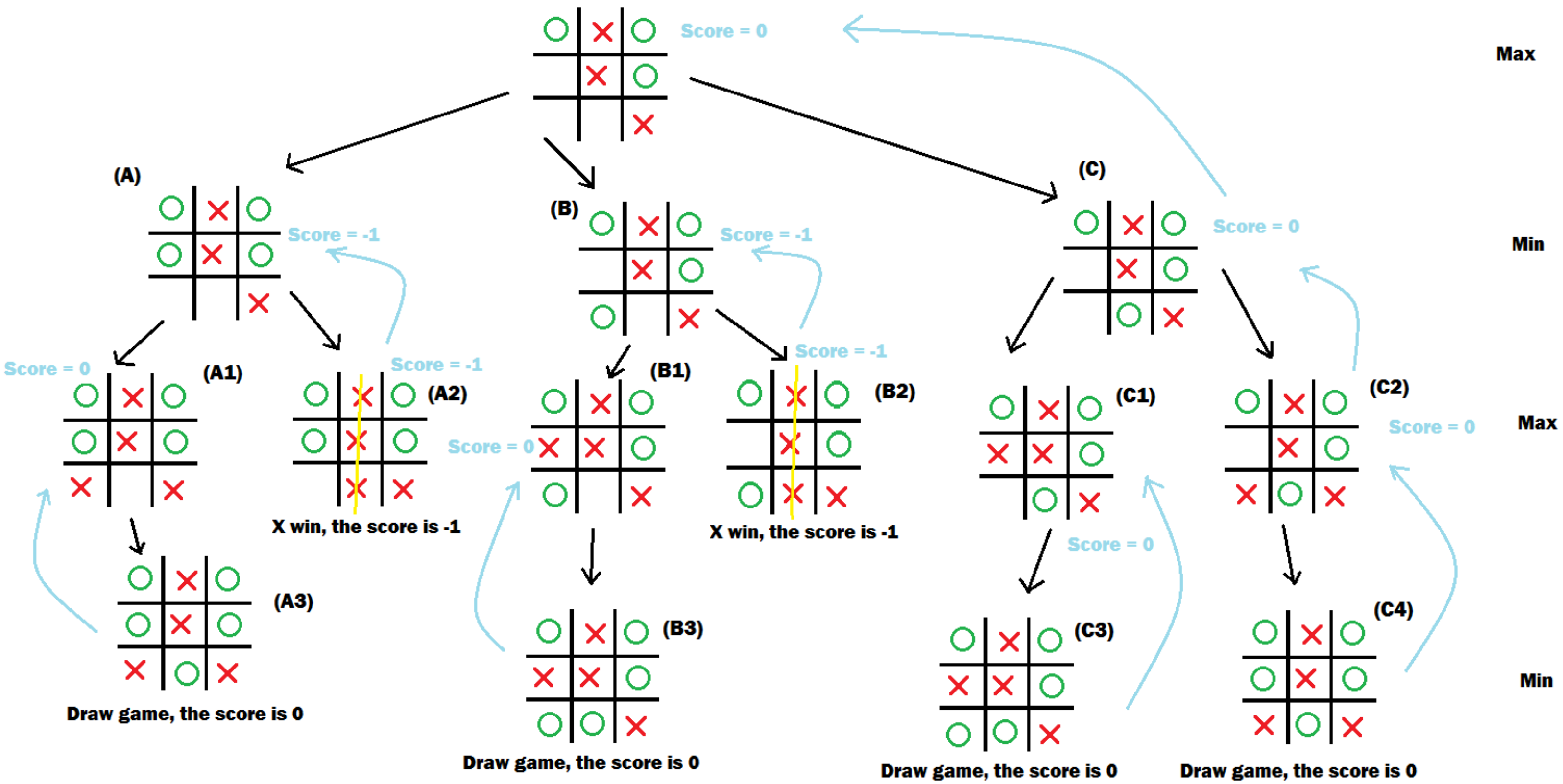
Figure 1. A (partial) game tree for the game of Tic-Tac-Toe.

# MiniMax

- Deterministic two –player zero – sum games
- Both player will play optimally.
- MAX-Player / MIN Player

$$\text{Minimax}(n) = \begin{cases} \text{Utility}(n) & \text{if } n \text{ is a leaf node} \\ \max_{s \in \text{Children}(n)} \text{Minimax}(s) & \text{if } n \text{ is a MAX node} \\ \min_{s \in \text{Children}(n)} \text{Minimax}(s) & \text{if } n \text{ is a MIN node} \end{cases}$$

# MiniMax




$$\text{Minimax}(n) = \begin{cases} \text{Utility}(n) & \text{if } n \text{ is a leaf node} \\ \max_{s \in \text{Children}(n)} \text{Minimax}(s) & \text{if } n \text{ is a MAX node} \\ \min_{s \in \text{Children}(n)} \text{Minimax}(s) & \text{if } n \text{ is a MIN node} \end{cases}$$

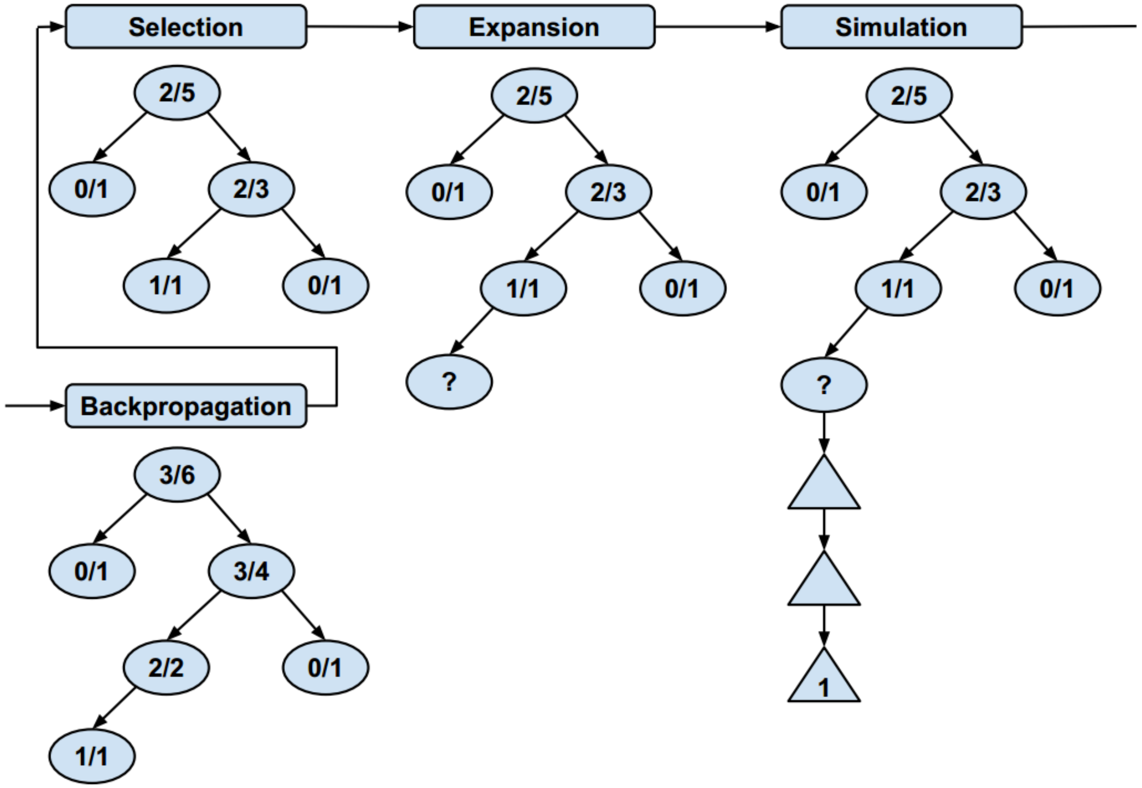
- MAX Player : “O –Player”
- MIN Player : “X-Player”

# Monte Carlo Tree Search

- Board games:
  - Hex
  - Go
  - Game of the Amazons
- Real-time video games:
  - Total War: Rome II

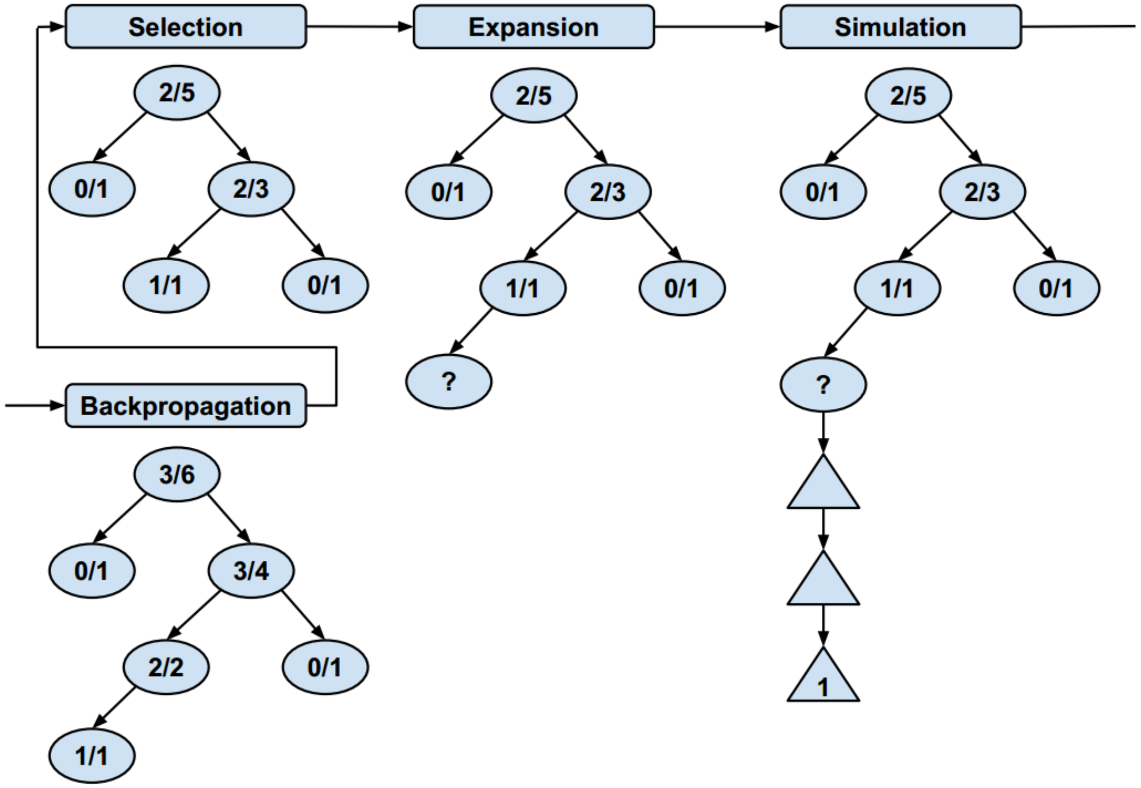

# MCTS

- Monte Carlo Tree Search(MCTS) is a method for finding optimal decisions in a given domain by taking random samples in the decision space and building a search tree according to the results.



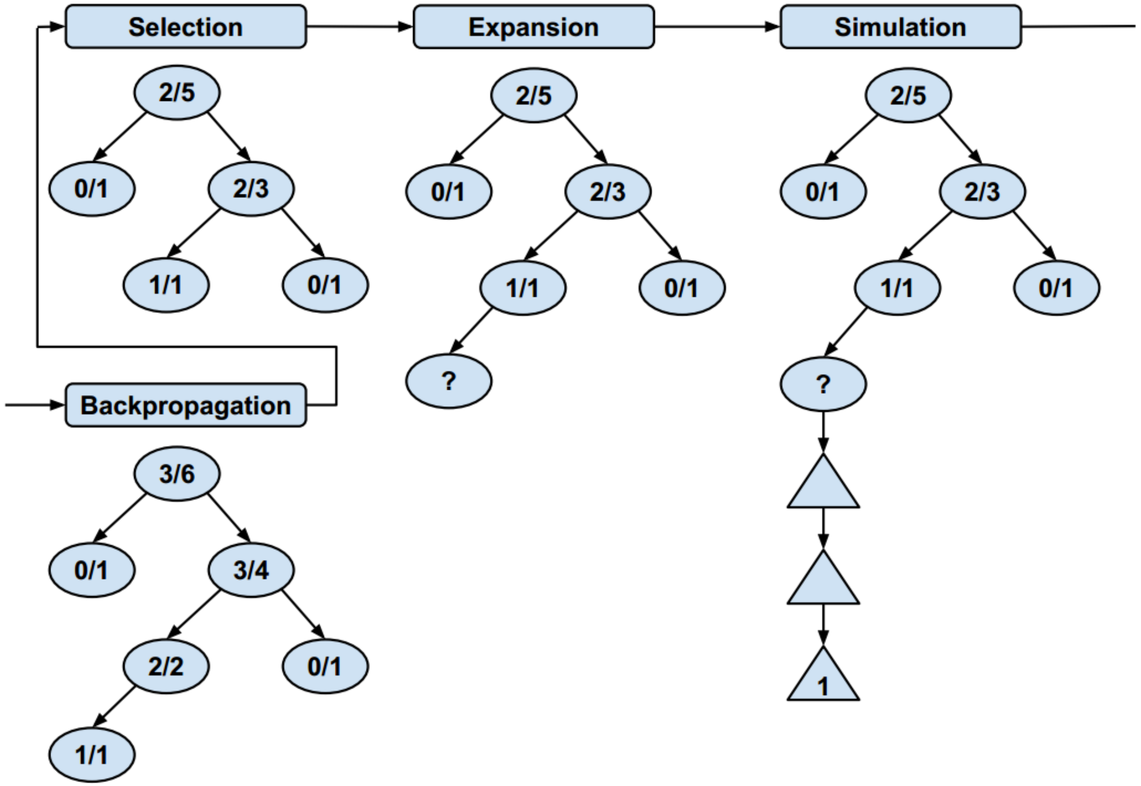
# MCTS –Step 1 Selection

- In the selection process, the MCTS algorithm traverses the current tree using a tree policy. A tree policy uses an evaluation function that prioritize nodes with the greatest estimated value.
- Each tree node stores the number of won/played playouts



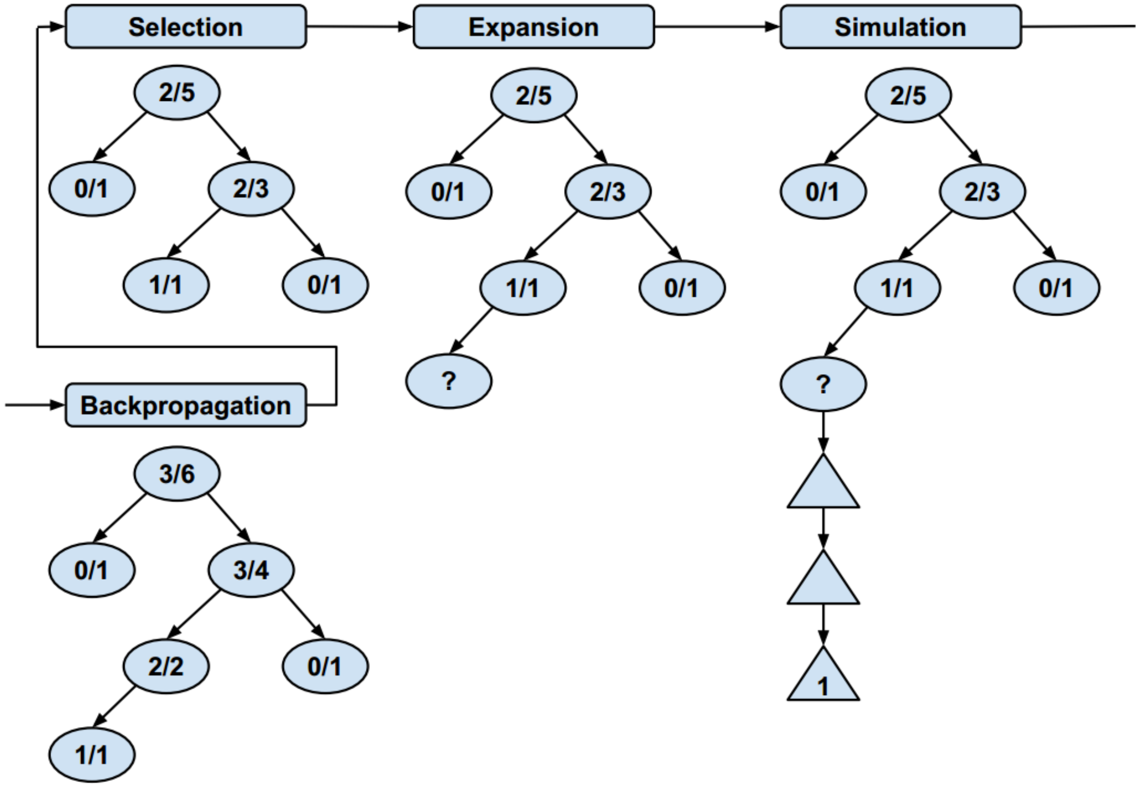
# MCTS –Step 2 Expansion

- In the expansion step, a new node is added to the tree as a child of the node reached in the selection step



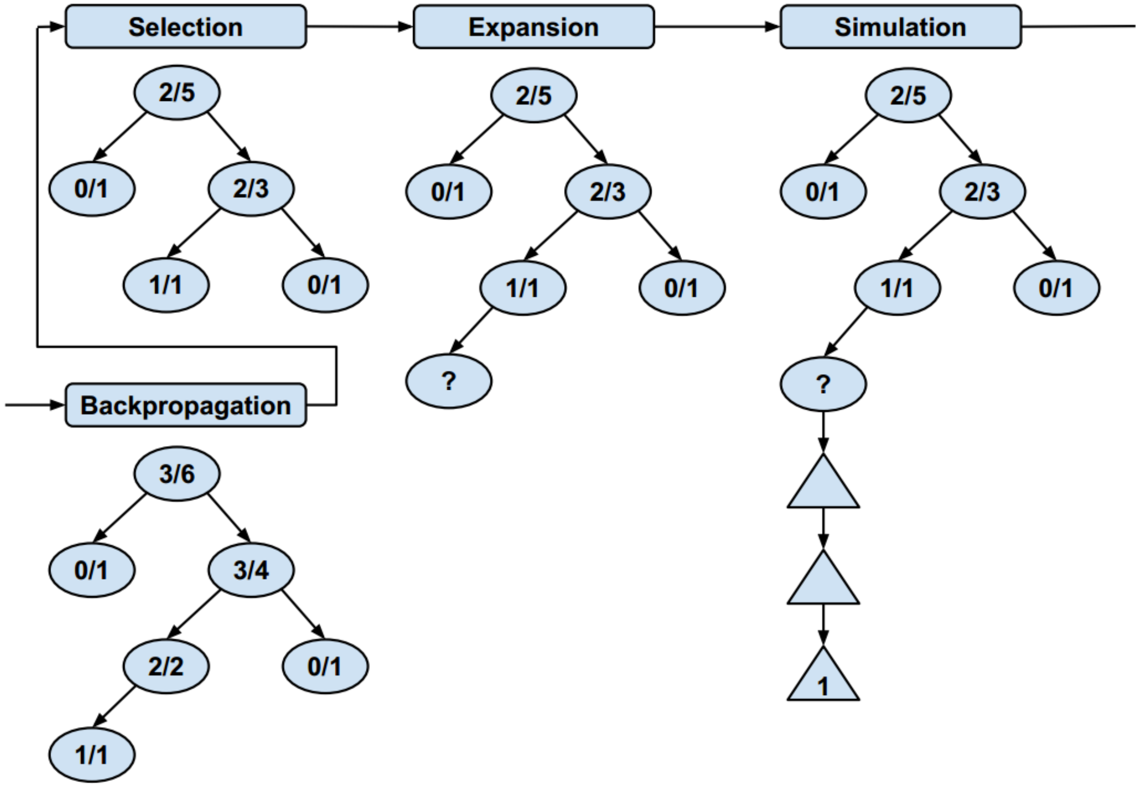
# MCTS –Step 3 Simulation

- In this step, a simulation is performed by choosing moves until either an end state or a predefined threshold is reached. Based on the result of the simulation, a the value of the newly added node is established.



# MCTS –Step 4 Backpropagation

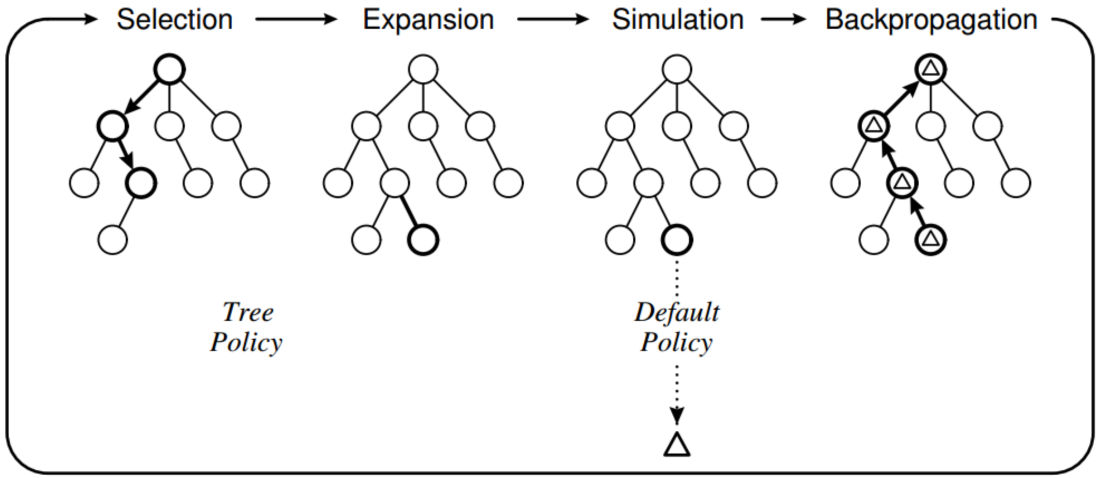
- Since the newly added node has been determined, the rest of the tree just be updated.



# Pseudo Code of MCTS

**Algorithm 1** General MCTS approach.

```
function MCTSSEARCH(s_0)
    create root node v_0 with state s_0
    while within computational budget do
        v_l ← TREEPOLICY(v_0)
        Δ ← DEFAULTPOLICY(s(v_l))
        BACKUP(v_l, Δ)
    return a(BESTCHILD(v_0))
```



# Upper Confidence Bond for Trees (UCT)

- Find balance between Exploration VS. Exploitation
- Exploration : explore unexplored areas of the tree.
- Exploitation: greedy, extends depth more than breadth.
- UCT balances exploration and exploitation  by giving relatively unexplored nodes and exploration bonus.

# UCT Equation

$$UCT(node) = \boxed{\frac{W(node)}{N(node)}} + c\sqrt{\frac{ln(N(parentNode))}{N(node)}}$$

N: the total number of simulations performed at that node
W: how many those simulations result a winning state
C: represents an exploration constants that is found experimentally.

# UCT Algorithm

**Algorithm 2** The UCT algorithm.

```
function UCTSEARCH(s_0)
    create root node v_0 with state s_0
    while within computational budget do
        v_l ← TREEPOLICY(v_0)
        Δ ← DEFAULTPOLICY(s(v_l))
        BACKUP(v_l, Δ)
    return a(BESTCHILD(v_0, 0))

function TREEPOLICY(v)
    while v is nonterminal do
        if v not fully expanded then
            return EXPAND(v)
        else
            v ← BESTCHILD(v, Cp)
    return v

function EXPAND(v)
    choose a ∈ untried actions from A(s(v))
    add a new child v' to v
        with s(v') = f(s(v), a)
        and a(v') = a
    return v'

function BESTCHILD(v, c)
    return argmax_{v' ∈ children of v} Q(v')/N(v') + c·sqrt(ln N(v) / N(v'))

function DEFAULTPOLICY(s)
    while s is non-terminal do
        choose a ∈ A(s) uniformly at random
        s ← f(s, a)
    return reward for state s

function BACKUP(v, Δ)
    while v is not null do
        N(v) ← N(v) + 1
        Q(v) ← Q(v) + Δ(v, p)
        v ← parent of v
```

The BESTCHILD return value:

$$\operatorname*{arg\,max}_{v' \in \text{children of } v} \frac{Q(v')}{N(v')} + c\sqrt{\frac{\ln N(v)}{N(v')}}$$

# Run the Program

- Visual Studio C++
- 3 by 3,  Traditional Tic-Tac –Toe Game
- Computational Budget : run 15000 times
- Modified Expansion step
- Future work (maybe) : Monte-Carlo Tree Search and Minimax Hybrids

# Reference


- Cameron Browne “A Survey of Monte Carlo Tree Search Methods”, *IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES, VOL. 4, NO. 1, MARCH 2012*
- B. Abramson, “Expected-Outcome: A General Model of Static Evaluation,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 12, pp.182 – 193, 1990.
- I. Althofer, “Monte Carlo Tree Search in Hex,” *IEEE Trans. Comp. Intell. AI Games*, vol. 2, no. 4, pp. 251–258, 2010.
- R. Agrawal, “Sample mean based index policies with zero (log n) regret for the multi-armed bandit problem,” *Adv. Appl. Prob.*, vol. 27, no. 4, pp. 1054–1078, 1995
- Hendrik Baier and Mark H.M. Winands,“Monte-Carlo Tree Search and Minimax Hybrids”, *Computational Intelligence in Games (CIG), 2013 IEEE Conference* .
- J. von Neumann and O. Morgenstern, *The Theory of Games and Economic Behavior*. Princeton: Princeton University Press, 1944.

# Question and Comments