Dynamic Programming and Reinforcement Learning Assignment 2

Soumyankar Mohapatra (2730063), Haohui Zhang(2722930)
Group 79

1 Introduction

In this assignment, the problems challenged us to find the optimum solution towards the tic-tac-toe game where we should arrive at a solution using the *Monte-Carlo Tree Search*(MCTS) algorithm. In the further sections, we will elaborate the algorithm and demonstrate the relevant results for the problem.

2 UCT

MCTS can be used along with different heuristics and is most-easily applied to games with an inherent tree structure. Thus, we try to develop an MCTS algorithm to solve this model-free reinforcement leaning problem.

The only precondition of MCTS is a generative model of the environment show below:

$$x', r \sim G(x, a). \tag{1}$$

In our environment, it meets this prerequisite, for taking each action, it will go to another state, however, the rewards are only given after the terminal state is reached. The winner is awarded 1 and the loser is rewarded -1.

The overall idea of MCTS is to estimate the highest expected return.

$$[V^*(x) = Q^*(x, a = \pi^*) = \mathbb{E}_{\pi^*}[r_o + \gamma r_1 + \gamma^2 r_2 + \cdots] \tag{2}$$

From the above equation, we calculate the highest expected return and use it to find the best policy.
We determined to use Upper Confidence Bound 1 applied to trees(UCT) to solve the problem, because UCT's efficient exploration of the tree enables to return rapidly a good value, and improve precision if more time is provided.
The UCT method comes from UCB(Upper Confidence Bounds) rule, which is a typical model for solving multi-armed bandit problem applying Hoeffding inequality.
Hoeffding inequality gives us, the exceedance probability of the sample mean($\bar{X}$) is exponentially bounded:

$$[P(\bar{X} \leq \mu - u) = P(\mu \geq \bar{X} + u) \leq e^{-2nu^2/L^2} \tag{3}$$

Where, μ is the actual mean($E(X_i)$) of random variables(X_i) such that $c \leq X_i \leq d$ where L = d - c.
Applying to estimation of bandit reward, the unknown real mean q(a) can be estimated by observed sample mean $Q_t(a)$:

$$[P(q(a) > Q_t(a) + U_t(a)) \leq e^{-2N_t(a)U_t(a)^2/L^2} := p(t) \tag{4}$$

Given exceedance probability (p(t)) to solve for upper limit $U_t(a)$, let exceedance probability go to zero over time and define $c = L/\sqrt{2}$, we get:

$$[U_t(a) = c\sqrt{\frac{\log t}{N_t(a)}} \tag{5}$$

Now, UCB1-Algorithm formula becomes to:

$$[a^*_{t+1} = \underset{a}{argmax}(Q_t(a) + c\sqrt{\frac{\log t}{N_t(a)}}) \tag{6}$$

The main difficulty in selecting child nodes for MCTS is to maintain a certain balance between the development of post-movement depth variants with a high average win rate and the exploration of moves with few simulations. To solve this, we generalize the UCB1-Algorithm into MCTS which becomes to UCT. Now, the formula is transformed to:

$$[a^*_{i+1} = \underset{a_i}{argmax}(\frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}}) \tag{7}$$

Where a is all selectable nodes; w_i is the number of wins of the node considered after the ith move; n_i represents the number of simulations for the child node; N_i defines total number of simulations for the parent node.
The first part of the above formula corresponds to exploitation; it is high for nodes with a high average winning percentage. The second part corresponds to exploration; high for nodes that are rarely simulated.

After determined the selection method, we randomly rollout from the leafs to get an estimate of the win/loose, backpropagate the result and update our model. After repeating a lot of times, we obtain the numerical results of the current node, find the policy which nearly converges to the optimal one and then move to next node. The algorithm is shown below.

Algorithm 1 UCT

$w \leftarrow 0$
$n \leftarrow 0$
$N \leftarrow 0$
for $t = 0, 1, \ldots, 999$ **do**
    for each $a \in \mathcal{A}$ **do**
        Compute the UCB bounds according to $B_i \leftarrow \frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}}$
    end for
    Select the node i_{t+1} as a child of node i_t that has the highest B value
    $i^*_{t+1} \in argmax_{i_t}(B_i)$
    $n_i \leftarrow n_i + 1$
    Roll-out to the terminal node using pure random selection
    if Player1 Win **then**
        $w_i \leftarrow w_i + 1$
    end if
    $N \leftarrow N + 1$
end for
return $\frac{w_i}{n_i}$

Algorithm 2 Random opponent

procedure UCT(node x)
    for each node x **do**
        if node x is a terminal node then **then**
            return heuristic value of node and policy
        end if
        if $a_{(p1)} \in \mathcal{A}_{p1}$ **then**
            for each $a \in \mathcal{A}_{p1}$ **do**
                $Q(x, a) = UCT(x)$
            end for
            $a \leftarrow argmax_{a \in \mathcal{A}_{p1}}(Q(x, a))$
        end if
        if $a_{(p2)} \in \mathcal{A}_{p2}$ **then**
            Select $a \in \mathcal{A}_{p2}$ randomly
        end if
    end for
end procedure

3 Maximin principle

In the previous process, we assume the opponent's selection is random and now we need to suppose the opponent plays optimally. In order to solve that, we follow the "maximin" principle. For that we goes first, we are also select the actions which will maximize our rewards and our opponent will try to minimize our rewards. In this section, we determined that our opponent also use the UCT to find the optimal policy. What's different, for the opponent, both the number of win for player 1(we) and player 2(the opponent) need to be recorded in UCT process. During the iteration of UCT selection, we will prefer to choose the node which has a high average winning percentage for player 2. When it comes to determine the action of the node for player 2, the action which results a minimum percentage of player 1 wining will be chosen. The algorithm is as follow:

Algorithm 3 Maximin - Optimal opponent

```
procedure UCT(node x)
    for each node x do
        if node x is a terminal node then then
            return heuristic value of node and policy
        end if
        if a_(p1) ∈ A_p1 then
            for each a ∈ A_p1 do
                Q(x, a) = UCT(x)
            end for
            a ← argmax_{a∈A_p1}(Q(x, a))
        end if
        if a_(p2) ∈ A_p2 then
            for each a ∈ A_p2 do
                Q(x, a) = UCT(x)
            end for
            a ← argmin_{a∈A_p2}(Q(x, a))
        end if
    end for
end procedure
```

4 Probability of Victory

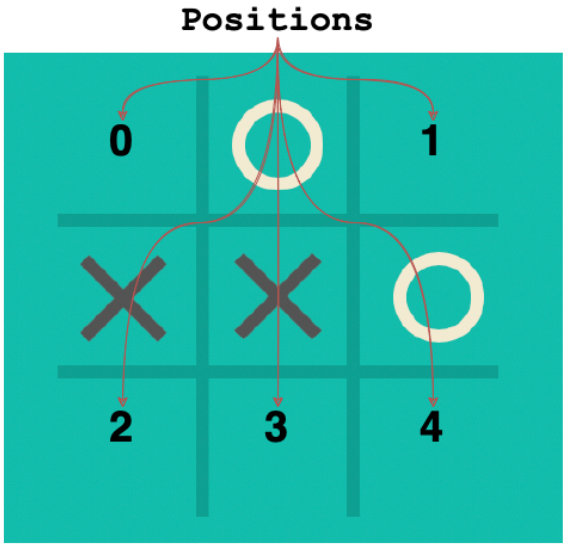

Figure 1: Representation of the problem state in the restricted condition

The problem challenges us to find the win probability starting in the restricted condition. We derive the probabilities by the tree structure that has been very meticulously described in the previous chapters. To verify the approach, we simulate the game over 1000 cases and derive the probability for all possible positions.

4.1 Random Opponent vs. MCTS UCT

In the table below, $MCTSX$ stands to say that the computer assumes player X and the opponent chooses a position randomly. The states $0, 1, 2, 3, 4$ represents the 5 open positions of the tic-tac-toe board.
In the above table we notice that using the UCT Approach we have a 100% win rate against a random opponent. The results are expected because the decision behind choosing a position is mathematically motivated. The reader may draw their attention towards the table on $position(3)$ which the UCT algorithm never chooses. The decision is supported through the theory above and it can be clearly seen as a poor decision in the figure.

	(0)	(1)	(2)	(3)	(4)
MCTS X Action Count	489	4	505	0	2
MCTS X Win Count	489	4	505	0	2
X Win Probability	100%	100%	100%	0%	100%
MCTS O Action Count	0	1000	0	0	0
MCTS O Win Count	0	1000	0	0	0
O Win Probability	0%	100%	0%	0%	0%

Table 1: Win Count to Action Count Ratio using UCT Approach

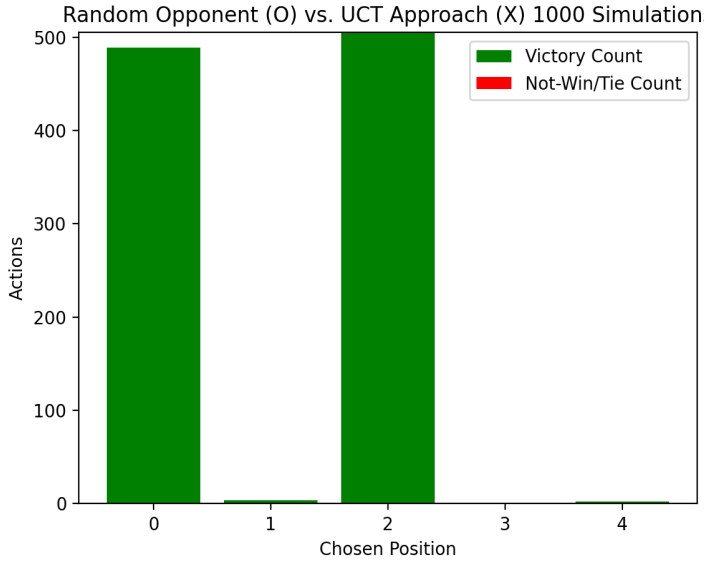

Figure 2: Bar Graph representing the Action to Win Rate for Random Opponent(O) against MCTS UCT(X)

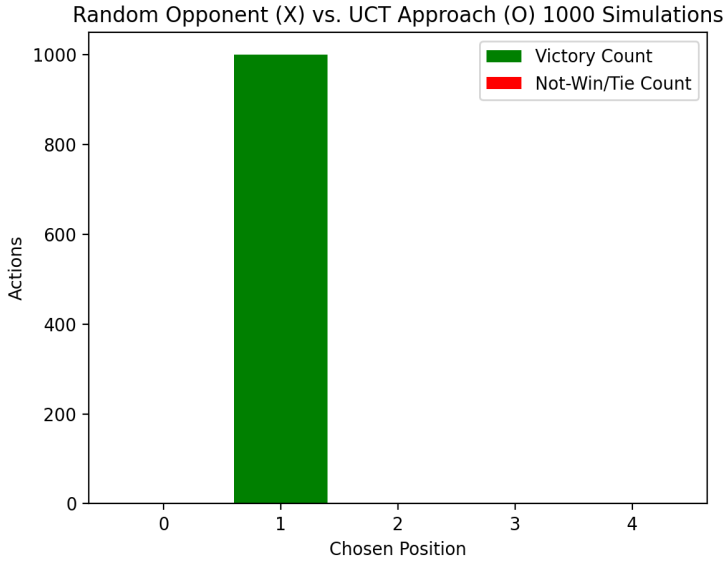

Figure 3: Bar Graph representing the Action to Win Rate for Random Opponent(X) against MCTS UCT(O)

4.2 Optimal Opponent vs. MCTS UCT

In this problem, we test the UCT approach against an optimal(smart) opponent. The table below shows the outcomes of the game depending upon the position the computer chooses.

	(0)	(1)	(2)	(3)	(4)
$MCTS\ X$ Action Count	495	5	498	0	2
$MCTS\ X$ Win Count	495	0	498	0	0
X Win Probability	100%	0%	100%	0%	0%
$MCTS\ O$ Action Count	0	1000	0	0	0
$MCTS\ O$ Win Count	0	1000	0	0	0
O Win Probability	0%	100%	0%	0%	0%

Table 2: Win Count to Action Count Ratio using UCT Approach against Optimal Opponent

Looking at the results above, we notice that the optimal opponent actually is able to beat the computer when the computer chooses the wrong position. This can be seen when the computer chooses $position(1)$ and loses 100% of the times.

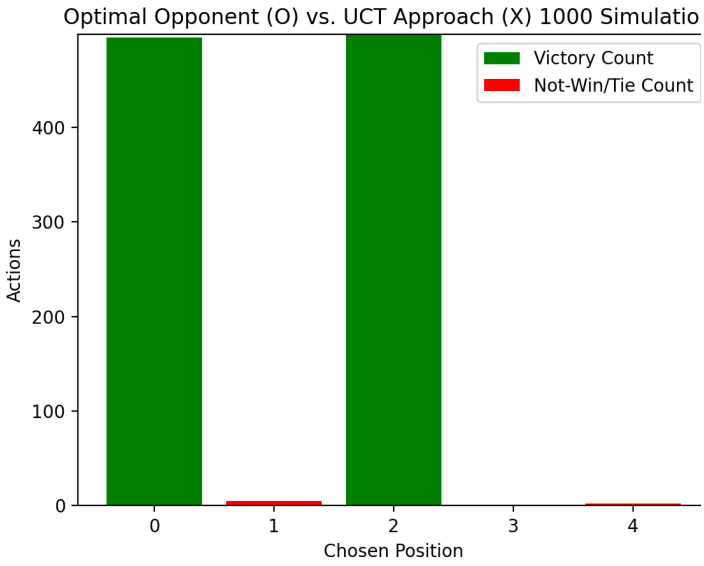

Figure 4: Bar Graph representing the Action to Win Rate for Optimal Opponent(O) against UCT(X)

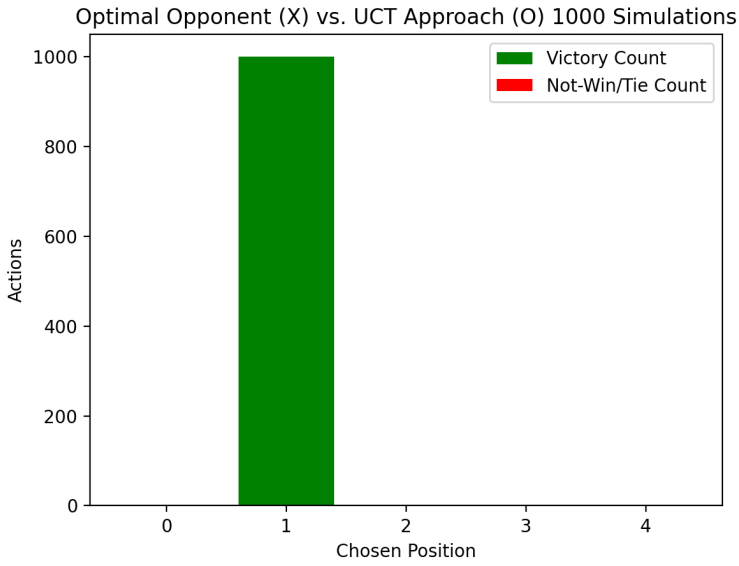

Figure 5: Bar Graph representing the Action to Win Rate for Optimal Opponent(X) against UCT(O)