Dynamic Programming & Reinforcement Learning 2020 **(Due: 13 dec 2020)**

Submission Assignment #3

Instructor: Ger Koole *Names:* Mick IJzer, *Netids:* 2552611

1 Introduction

The goal of this assignment is to implement a Monte-Carlo Tree Search (MCTS) algorithm that can reliably beat a random agent in a game of tic-tac-toe. MCTS is a search algorithm that finds the optimal policy by searching a decision tree. The steps in MCTS are; select a node based on a policy, roll out with random decisions to get the reward, and finally update the obtained rewards from the nodes that were passed. Given is that the reward for winning is $r = 1$, for a draw $r = 0$ and for losing $r = -1$. For this assignment three starting states will be considered and three exploration/exploitation strategies (i.e. random, UCB, and ϵ-greedy) will be evaluated. Furthermore, the performance of the strategies for Player 1 (i.e. crosses) will be evaluated for different strategies for Player 2 (i.e. circles). To make it easier to refer to certain positions on the board, the positions will be numbered from 1 through 9 starting at the top left.

The three different starting states are shown in figure 1. For starting state 1 the optimal policy for Player 1 is easy to deduce. Let us assume that Player 2 randomly selects a position and Player 1 follows the optimal policy for that state. In that case the probability of victory for Player 1 given $action = position$ is given in equations 1.1, 1.2, 1.3, and 1.4. For the other two starting states the optimal policy is harder to deduce. However, for starting state 2 the optimal policy for Player 1 would result in a guaranteed victory, independent of the actions of Player 2.

$$P(victory|action = 6) = P(victory|action = 9) = 1 \tag{1.1}$$

$$P(victory|action = 1) = \frac{3}{4} + (\frac{1}{4} * \frac{1}{2}) = \frac{7}{8} \tag{1.2}$$

$$P(victory|action = 4) = \frac{3}{4} \tag{1.3}$$

$$P(victory|action = 8) = 4 * (\frac{1}{4} * \frac{1}{2}) = \frac{1}{2} \tag{1.4}$$

As explained earlier, the first step in MCTS is to select a node from which to carry out the rollout. This selection can be done randomly or by using a exploration/exploitation strategy. In addition to the Random strategy, the Upper Confidence Bound (UCB) and ϵ-greedy strategies will be used. For the Random strategy it is obvious that random choices will be made at each decision to select the starting node. The ϵ-greedy strategy will almost always choose the optimal position (based on historic rewards). However, with probability ϵ a random position will be chosen. The chosen value for ϵ indicates the level of exploration. Lastly, the UCB strategy will select the node with the highest UCB-value. the UCB-value for each node can be computed using equation 1.4, where k is the number of times the specific node in the tree has been visited, t the total number of iterations, and c a constant. Equation 1.5 boils down to the average reward that has been obtained plus a confidence bound that is based on the number of iterations and the number of times the node has been visited. The confidence bound becomes larger if the state is not visited for a long time. So, the longer a node has not been visited the more likely it is to be chosen the next time. Also, a larger value for c will lead to a higher value for the confidence bound, which results in more exploration.

$$UCB = \frac{\sum_{i=0}^{k} r_i}{k} + c * \sqrt{\frac{log(t)}{2 * k}} \tag{1.5}$$

2 Methodology

First, the tic-tac-toe game was implemented. The game contained hard-coded win conditions, a state representation, the possible actions, and the winner (if any). The chosen state representation was the order in which the positions on the board were chosen. Every other position in the state representation were the chosen positions for Player 1. For example, in the state representation $[3, 2, 5, 7]$ Player 1 has chosen positions 3 and 5, and Player 2 has chosen positions 2 and 7. Based on the state representation it is simple to deduce the possible options, namely everything in the range 1 through 9 that isn't in the state representation. The MCTS-algorithm is implemented according to the described steps in section 1.

The experiment itself was carried out using the implementation of the game and of the MCTS-algorithm. For all three starting states the same strategies of Player 1 and Player 2 were used. The possible strategy-combinations were; Random/Random, UCB/Random, ϵ-greedy/Random, Random/UCB, UCB/UCB, where the first strategy is of Player 1 and the second of Player 2. This means that in the end five strategy-combinations and three starting states were simulated. An important note is that the number of possible trajectories for starting state 1 is $5! = 120$, while for the other two this is $7! = 5.040$ and $9! = 362.880$ respectively. Therefore, the number of iterations will be increased for the more 'complex' starting states. The exact set-up is schematically shown in table 1.

3 Results and Discussion

The experiments were carried out according to the setup as described in table 1. The learning process is visualized in figure 2. It is important to note that the lines that are shown in the figure reflect the moving average reward of the last 100 iterations. This is done to improve the readability of the figures. For starting state 1, it can be seen that any non-Random policy of Player 1 (i.e. the purple, yellow and green lines) will result in an average reward of 1, independent of the policy of Player 2. This is not the case when Player 1 follows a Random policy (i.e. the blue and red lines). However, since the number of iterations is higher than the possible number of trajectories, the stored values for each node do reflect the optimal policy. So if Player 1 were to switch to a greedy strategy they would follow the optimal policy and are guaranteed to win. It is possible for Player 2 to win if Player 2 has UCB as their strategy and Player 1 has a Random strategy. However, even if Player 1 has a Random strategy it is still difficult for Player 2 to win.

The figure for Starting State 2 shows a similar pattern. If Player 1 doesn't use a Random strategy, Player 1 should always win. This is reflected by the lines for the following strategy-combinations UCB/Random, Greedy/Random, UCB/UCB. With this starting state it is easier for Player 2 to win if Player 1 has a Random strategy. This makes sense since the number of possible trajectories is a lot higher. The most interesting figure is the one for Starting State 3; the empty board. Both strategy combinations UCB/Random and Greedy/Random will most likely result in Player 1 winning. Random/UCB will result in Player 2 winning. However, the result of UCB/UCB is almost always a draw. This makes sense since Player 2 can always force a draw if Player 1 plays their optimal policy. For the Random/Random strategy combination the pattern for all three starting states is similar; Player 1 is more likely to win. This is probably due to the fact that Player 1 goes first and will have five positions they can select compared to only four positions for Player 2.

The conclusion that can be drawn from this experiment is that MCTS can be used to create an agent that is able to reliably win from a random agent for all starting states. Especially the results from Starting State 3 are remarkable, since ratio of possible trajectories to number of iterations is a lot lower. This means that there is no way for the agent to have seen all possible trajectories. Yet the agent is still able to find an optimal policy that is able to win against a random agent.

Appendix

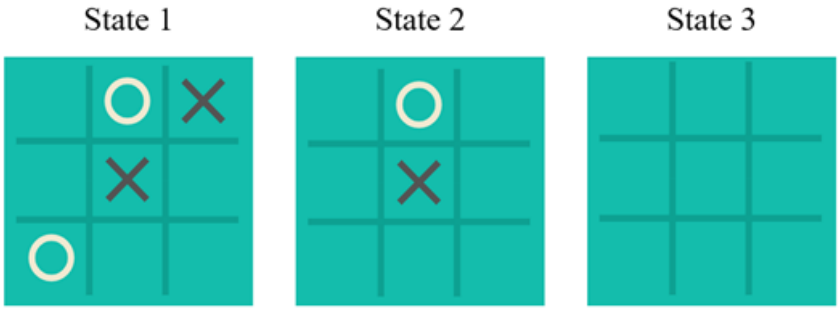

Figure 1: The three starting states.

	Starting State 1	**Starting State 2**	**Starting State 3**
Possible Trajectories	120	5.040	362.880
Iterations	1.500	2.500	7.500
Strategy Combinations	5	5	5
c (of UCB)	1	1	1
ϵ (of ϵ-greedy)	0.05	0.05	0.05

Table 1: Experimental Setup

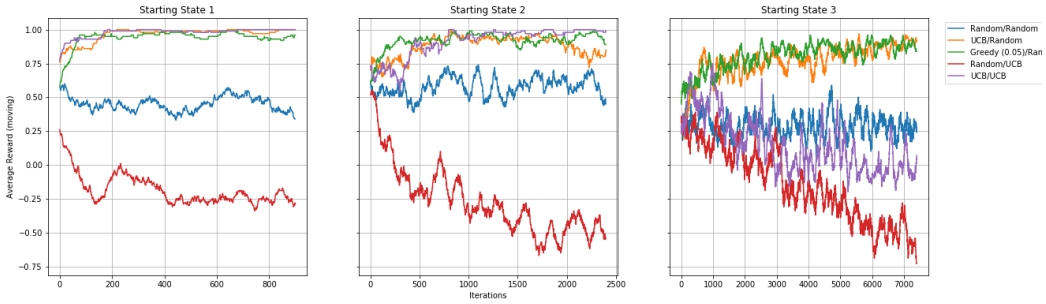

Figure 2: Moving average of the accumulated rewards for the three starting states.