# Artificial Intelligence(CS-F405): Assignment 2 report

Akhilesh Adithya

December 4, 2021

## 1 Introduction

This report reports the results obtained while performing assignment 2. This assignment required us to implement the game "Connect 4" in python and then create AI algorithms to beat a player using either MCTS or Q-learning. The assignment report is divided into the following parts.

(a) The Monte Carlo Tree Search [ $MC_{200}$ v/s $MC_{40}$ ]

(b) Implement and train Q-learning against $MN_N$

(c) Train Q-learning so that it beats $MC_N$

# 2 $MC_{200}$ v/s $MC_{40}$

## 2.1 Architecture

A slightly modified version of MCTS was used in this assignment. The modifications done and the rationale behind them are discussed in the following subsections

### 2.1.1 Rewards

The rewards system was revamped and was made to use the following rewards.

- 1 in case of a victory
- -10 in case of a draw
- -100 in case of a loss

This was changed as in our case, the worst possible outcome would be a loss. A draw would be desirable when compared to a loss, but still undesirable when compared to a victory. Hence we follow an optimistic strategy were the initial value of each state is set at 5, but a draw or a loss would change this value during backprop.

### 2.1.2 UCT function

The UCT function has been changed from

$$UCT = X + C * \sqrt{\frac{log(n)}{n_j}}$$

Where X is the win-visit ratio, C is a constant, n is the number of times the parent node was visited, and $n_j$ was the number of times the child node was visited. To -

$$UCT = \frac{Reward}{n_j} + C * \sqrt{\frac{log(n)}{n_j}}$$

Where reward is the reward given. [see Sec 2.1.1]

## 2.2 Results

### 2.2.1 Result when played for 100 games

The results derived when a $MC_{40}$ alsorithm is pitted against a $MC_{200}$ in a $6row \times 5column$ game for a 100 times, where the access given to play the first move is alternated every game. [50 games for each algo]

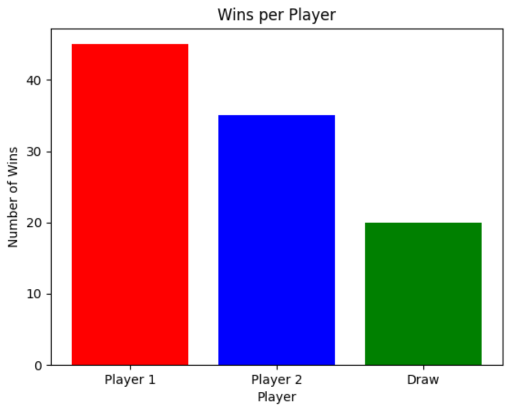


Figure 1: Count of wins based on which player started first

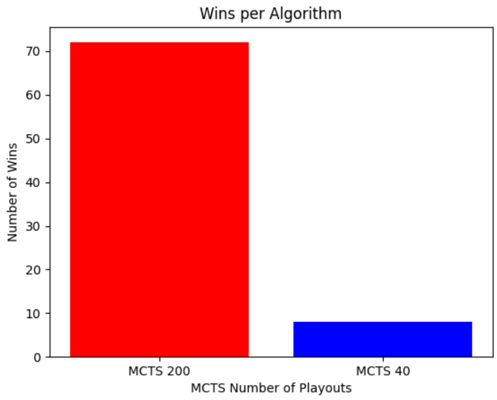


Figure 2: Count of wins based on which version of the MCTS algorithm started first. Constant C is fixed at $\sqrt{2}$ to follow the UCB1 formula

From these results [Figure 1], we can safely conclude that playing first has a distinct advantage over playing second. This can be assumed to be the reason as playing first gives initiative to the algorithm to perform and less focus needs to be placed on preventing the opponent from winning, and can give a considerable advantage.

We can also conclude from these results [Figure 2], that $MC_{200}$ has a definitive advantage over the $MC_{40}$ algorithm. In this case. we see that $MC_{200}$ has 72 wins and the $MC_{200}$ algorithm has 8 wins. Even though $MC_{200}$ is a better algorithm as it can simulate more steps, we see these results due to the inherent stochastic nature of the $\epsilon - greedy$ algorithm.

## 2.3 Impact of Choice of parameter on performance of $MC_{200}$

The most important parameter that affects the performance of the $MC_{200}$ algorithm is the constant **C**, in the UCT[Upper Confidence bounds on Trees] formula. The constant C clearly corresponds to the amount of exploration allowed to the algorithm. And as $MC_{200}$ has more playouts when compared to $MC_{40}$, the $MC_{200}$ algorithm would get a huge boost as the value of C increases. This also results in relatively poorer performance of the $MC_{40}$ algorithm.

This assumption is further confirmed with tests. The following is is a graph plotted with the value of C as the x-axis and the percentage of wins in a 100 matches against $MC_{40}$ algorithm with $MC_{200}$ always starting first.

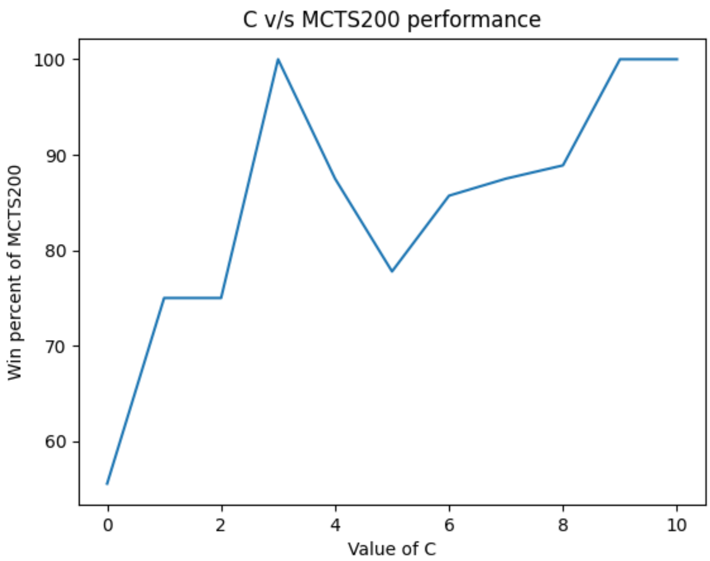


Figure 3: Percentage of wings against $MC_{40}$ algo v/s Values of constant C

Here, the $MC_{200}$ was made to play against the $MC_{40}$ algorithm 10 times for varying values of the confidence parameter(C). To account for the irregularity due to the stochastic nature of the entire game, the algorithms was pitted against 10 times per value of C. We can attribute to the slight irregularity of the upward slope due to the stochastic nature of the entire process, but we see a definite correlation between the value of C and the percentage of wins against the $MC_{40}$ algorithm. There might be a slight bias towards the results as the $MC_{200}$ is always set to start first, but the bias is negligible as seen in Figure 1.

### 2.4 Conclusion and Alternative approaches

We can safely conclude that the $MC_{200}$ algorithm performs better than the $MC_{40}$ algorithm. We can also conclude that the player who starts first has a definitive advantage. We can also conclude that the parameter C in the UCT formula directly impacts the performance of the $MC_{200}$.

Some other alternative approaches that could have been tried out but weren't done in this paper are now discussed. The **values of rewards** could be tuned as a hyper parameter for improving the $MC_{200}$ algorithm. Theoretically, changing the rewards such that -

- Reward for winning = 10
- Reward for losing = -10
- Reward for draw = 0

would ensure that $MC_{200}$ would win more as more simulations would lead to a higher probability for winning. But changing the rewards would

But this has not been done in this paper as this would ruin the optimistic start approach followed in this paper. Another thing that could have been tried is that the UCT formula could have been replaced by either the RAVE or the PUCT formula as they have been tested out to be more robust and faster for the Monte Carlo Tree system.

# 3 Implement and train Q-learning against $MN_N$

In this section, the first part is the original idea implemented before the question was modified. Or at least an attempt to implement the question before it was edited. The second part deals with the submitted version of the Q-learning algorithm. In the last part, the conclusions are drawn and alternative methods are discussed

## 3.1 Original Idea

Originally, the Q learning algorithm was trained on a $5 \times 6$ connect 4 game board. This Q learning algorithm was trained against a $MC_{200}$ algorithm. The results were sub par and Q learning algorithm couldn't beat the $MC_{200}$ algorithm even after a 1000 rounds of training.

```
Final State:
[[0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 1 0]
 [2 0 0 1 0]
 [2 0 0 1 0]
 [2 0 0 1 0]]
The player who won is: 1
```

Figure 4: Result of the first game during the 1000 rounds of training against $MC_{200}$ algo

```
Final State:
[[2 2 1 0 0]
 [1 1 2 0 1]
 [2 1 1 0 1]
 [2 2 2 0 1]
 [2 1 1 2 1]
 [1 1 2 2 2]]
The player who won is: 1
```

Figure 5: Result of the last game during the 1000 rounds of training against $MC_{200}$ algo

We clearly see that even though Q learning cannot beat $MC_{200}$ algo, it still improves and learns to perform better. But this is clearly not enough to get the Q learning to win. There were a few ways I thought of improving this like,

- Changing the values of $\alpha, \epsilon, \beta$
- Changing the rewards
- Increasing the training time
- Start training Q learning first against a random player, then a $MC_{20}$ and finally against $MC_{200}$ so that the learning is meaningful

But due to time constraints, I was not able to implement these. But I do plan to pursue this in the future.

## 3.2 Simplified Connect 4 game

In this section, we start by simplifying the connect 4 game. The connect 4 board is reduced from $5 \times 6$ to $5 \times 2$. Essentially, we're crippling the game from horizontal, vertical and diagonal win states to just a horizontal win state. This would simplify the game space and make it easier for the Q-learning algorithm to converge.

### 3.2.1 Against $MC_0$ (Random player)

In this section, we discuss the results that we get when the Q learning algorithm is played against the $MC_0$ [MC0 ensures it does random moves all the time].

Here, the Q Learning algorithm was made to play as player 2 against the $MC_0$ algorithm as player 1. The Q-learning algorithm was trained for 1000 epochs and we see how it fares against this system. The results are as follows -

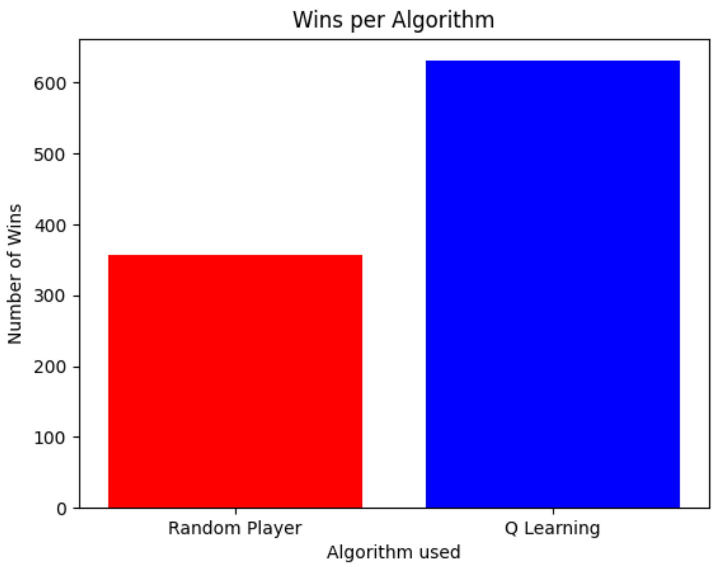


Figure 6: Count of wins based on which algorithm was used

We see that the Q learning algorithm learns and can beat a random player with a considerable margin.

### 3.2.2 Against $MC_1$ (With just 1 simulation)

In this section, we discuss the results that we get when the Q learning algorithm is played against the $MC_1$ algorithm.

Here, the Q Learning algorithm was made to play as player 2 against the $MC_1$ algorithm as player 1. The Q-learning algorithm was trained for 1000 epochs and we see how it fares against this system. The results are as follows -

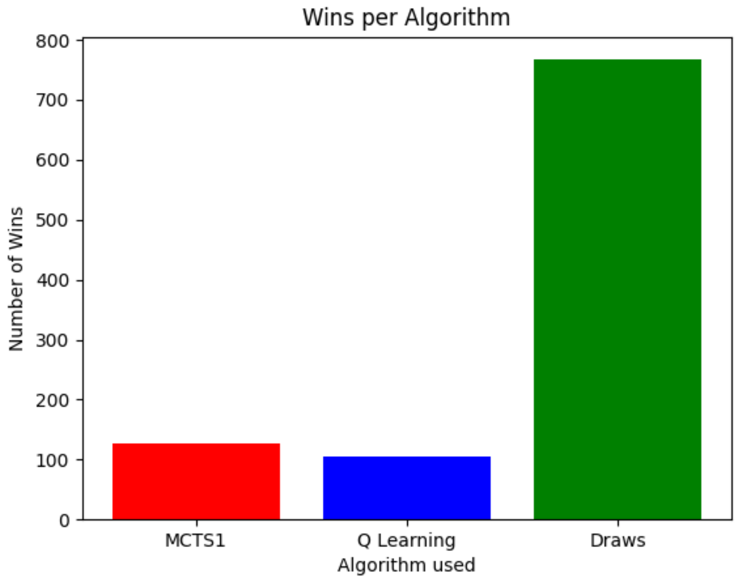


Figure 7: Count of wins based on which algorithm was used

We see that the $MC_1$ algorithm slightly outperforms the Q-learning algorithm. But this is to be expected as playing as player 1 has an added advantage to itself. This can also be attributed to how well MCTS performs for games such as monte carlo which has huge game spaces.

#### 3.2.3 Against $MC_{10}$ (With simulation of upto 10 playouts)

In this section, we discuss the results that we get when the Q learning algorithm is played against the $MC_{10}$ algorithm.

Here, the Q Learning algorithm was made to play as player 2 against the $MC_2$ algorithm as player 1. The Q-learning algorithm was trained for 1000 epochs and we see how it fares against this system. The results are as follows -

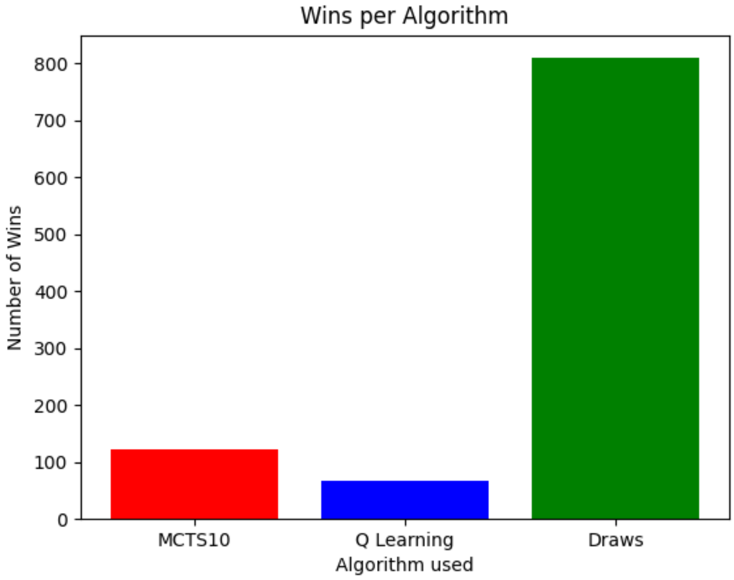


Figure 8: Count of wins based on which algorithm was used

We see that the $MC_{10}$ algorithm outperforms the Q-learning algorithm. But this is to be expected as playing as player 1 has an added advantage to itself. This can also be attributed to how well MCTS performs for games such as connect 4 which has a moderate game space.

When compared with the $MC_1$ algorithm, we see that the number of wins that $MC_{10}$ algorithm has is pretty much the same, but the number of wins that the Q learning algorithm has decreases significantly.

# 4 Train Q-learning so that it beats $MC_N$

The only value of N that Q Learning was able to beat was for $N = 0$. As these results were sub par, some ideas that were tried out are listed

## 4.1 Ideas Implemented

- Tuning values of $\alpha, \gamma$ and $\epsilon$
- Increasing training time
- Start training with a random player, then move on to opponents with higher intelligence

## 4.2 Results

Following all the steps mentioned above, the values of $\alpha, \gamma and \epsilon$ were set to 0.5, 0.9 and 0.1 respectively. The algorithm was trained for 10000 iterations which took around 2 hours of training. In the end, the final Q values were saved as the .dat.gz file required. Under these circumstances, we saw that $MC_1$ loses against Q learning. But Q learning is not able to win against any other intelligent opponent. This is an improvement when compared against the original Q learning algorithm, but is still not satisfactory.

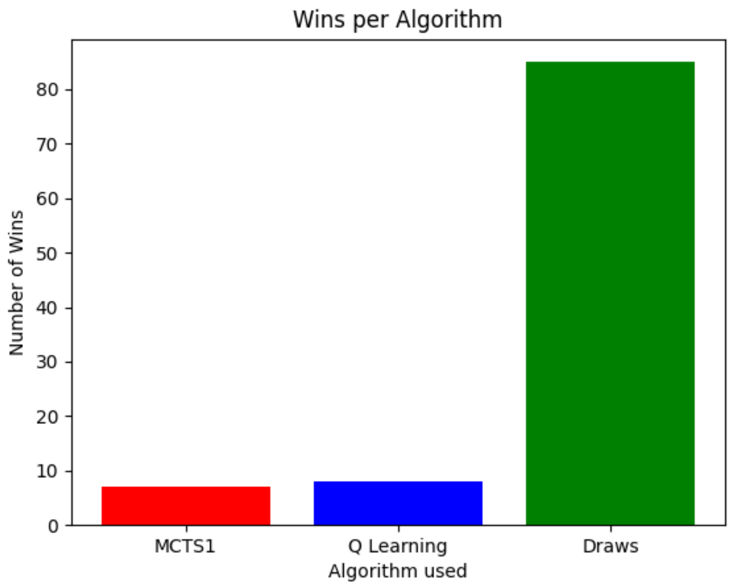


Figure 9: Count of wins based on which algorithm was used