# Reinforcement Learning with Tic-Tac-Toe

**Eric Chen (4033) and Nick Thompson (4033)**

## Abstract

We investigated the game of Tic-Tac-Toe and designed two agents utilizing two different machine learning models to create a potentially near optimal player. In our experiments we compared the effectiveness of the two agents as well analyzing and testing their training requirements to reach optimal states. Overall we found that between the two models we used (Q-learning and Sarsa) Sarsa proved to be the superior learning model for the game of Tic-Tac-Toe.

## Introduction

Tic-Tac-Toe is a common childrens' game in which two people compete with one another and take turns making marks on a (typically) 3 by 3 board. The player who successfully manages to place their symbols either horizontally, vertically, or diagonally is deemed the winner. As such, we sought out to explore various learning models and design a high performance tic-tac-toe agent. In the Tic-Tac-Toe environment that we decided to analyze, there are 9 total cells available to be marked, and a total of 765 essentially different board states (rotations and reflections not counted). Furthermore, only 138 of these 765 result in a terminal state, meaning a win, loss, or draw for a player(s). Specifically, 91 of these states result in a win for player one--typically X--44 of these states result in a win for player two--typically O--and 3 of these states result in a draw (Schaefer 2002).

### Literature Review

In our first reference (Steeg, Drugan, and Wiering, 2015), the researchers analyzed a game similar to tic-tac-toe using TD-learning: Tic-Tac-Toe 3D – though it is much more complex. One novelty in their approach was the usage of a benchmark player. This player's policy consisted of randomizing all possible moves, and following a rule-based approach to determine its next move. We decided to implement a similar agent, that upon randomizing all possible moves, would look for any possible winning moves before simply just taking a random move. Furthermore, the researchers found that by testing their agent against their benchmark player, there was a substantial improvement in win percentage when in comparison to the agent training against itself (van de Steeg, et. al, 2015). As such, this presents us with another experiment for us to analyze in our own project.

According to Sutton and Barto in 'Reinforcement Learning, an Introduction', Q-learning is an 'off-policy' TD control method, meaning that Q directly approximates $q_*$, independent of the followed policy (Sutton, et. al. 2018). The difference between Sarsa and Q-learning methods are characterized by the given example 6.6 of cliff walking. In that, Q-learning has worse online performance than Sarsa which takes a safer path to learn the optimal policy. For us, this would mean that the Sarsa and Q-learning agent would both achieve an optimal policy in tic-tac-toe, but the Q-learning agent would achieve it faster while the Sarsa agent would lose less games as it trained.

## Experiment Review

### Learning Methods

In approaching this project, we sought out to use Q-learning and Sarsa. Upon our initial implementation, we realized that our TD-learning implementation would be ineffective at selecting actions within the state-space as they are not tracked within a table, as such, we decided to utilize Sarsa instead. Nick worked to implement the Q-learning model, while Eric worked to implement the Sarsa model.

Q-learning is the off-policy TD control algorithm defined by:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$

Sarsa is the on-policy TD control algorithm defined by:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right]$$

We chose Sarsa and Q-Learning methods as they are both TD control algorithms, hence we could use them to evaluate and find the 'optimal policy' for tic-tac-toe to win or draw any game. In our implementations, we chose to build upon an existing Tic-Tac-Toe library for python designed by Kirill Bobyrev found on Github (Bobyrev 2018).

### Hypotheses

Firstly, we hypothesized that we could design a tic-tac-toe agent utilizing Q-learning or Sarsa algorithms that would be able to always win or draw a match.

Secondly, we hypothesized that Q-learning will take less training iterations to achieve an optimal policy than Sarsa.

Based on the first reference, our third hypothesis is that training the agents against a randomized, rule-based agent (benchmark player) would produce better trained agents that would have a higher win percentage and require less training iterations.

Finally, we hypothesized that when training against the benchmark player, Q-learning will result in faster learning as it is more likely to take greedy moves in comparison to Sarsa.

## Experiments

For our first experiment, we designed a Q-learning agent with a learning rate α and discount rate γ of 0.9 and 0.95, respectively. In this model, the agent records a history of the moves it makes and accordingly applies the reward at the game's termination. Furthermore, for this experiment, we trained the agent by having the agent play itself and allowed the agent to train over 8000 iterations.

Following this up, another experiment we conducted was to train the Q-learning agent against a randomized, rule-based player rather than itself. During this experiment, we kept the learning rate and discount rate consistent with our previous experiment.

For our third experiment, we utilized a Sarsa learning model to design an agent with a learning rate α of 0.9, and a discount rate γ of 0.95.

Similar to Q-learning, we also conducted a fourth experiment in which we trained the Sarsa agent up against our randomized, rule-based player instead of itself. Much like before, we kept all the rates consistent with our previous experiment.

# Analysis

## Q-learning

In our first experiment in which we designed a Q-learning agent and trained it against itself, we were able to achieve the following game outcomes during the training iterations:

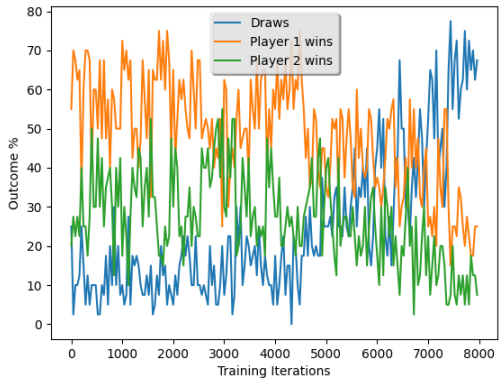


Figure 1: Q-learning training against itself.

As we can see, as the agent learns over the various iterations, it eventually reaches a point in which most if not all the games result in a draw. Using this trained agent, we then compare it to our benchmark player, which makes moves by randomizing the set of all possible moves and returning any winning moves. If no moves are winning, then a random move is given. Pitting our agent against this player, we gathered the following:

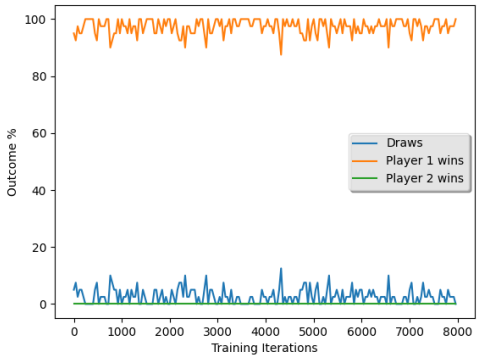


Figure 2: Q-learning vs Q-learning benchmark

Looking at this graph, we can see that our trained agent managed to win most if not all the games against our benchmark player. Furthermore, matches that were not won were rather drawn – primarily indicated by the 0% win percentage of player 2 (benchmark player). As such, this supports our first hypothesis that a properly trained agent would be able to never lose a game of tic-tac-toe.

For our second experiment, we looked at how effective it would be to train our Q-learning agent against the benchmark player rather than against itself. The training of the

experiment gave us the following:

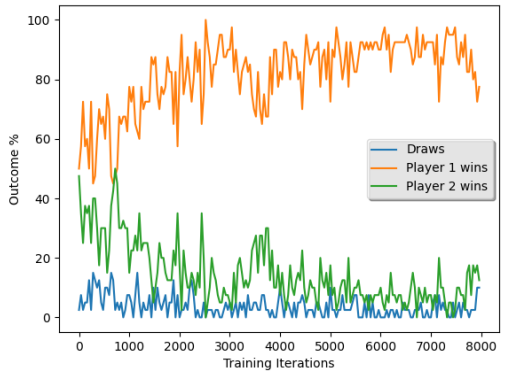


Figure 3: Q-learning training against benchmark.

As we can see, this training graph differs drastically from when the agent trained against itself as in this experiment, only player 1 (the Q-learning agent) is learning--further indicated by the rising win percentage. After training the agent against the benchmark player, we then ran another set of iterations to evaluate the effectiveness of the training. The results as follow:

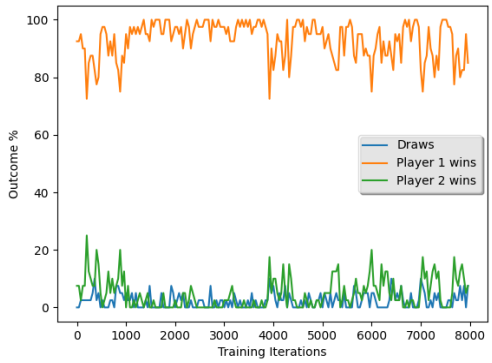


Figure 4: Q-learning vs benchmark benchmark.

Looking at this graph, we see that the results of the training were not as effective as when the agent trained against itself. Though the results are still heavily in favor of player 1 (Q-learning agent), the benchmark player still managed to take some games and as such, does not support our third hypothesis that training against the benchmark player would be more effective.

## Sarsa

In implementing our Sarsa model, we were able to find the following outcome upon training the model with itself:

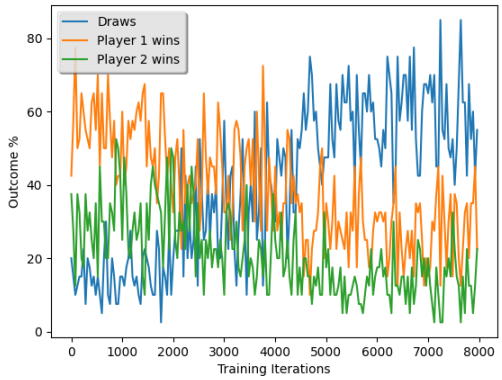


Figure 6: Sarsa training against itself

It can be seen that over time, as the agent improved, the amount of draws gradually increased. As such, we can then compare the trained model with our benchmark player, giving us the following results:

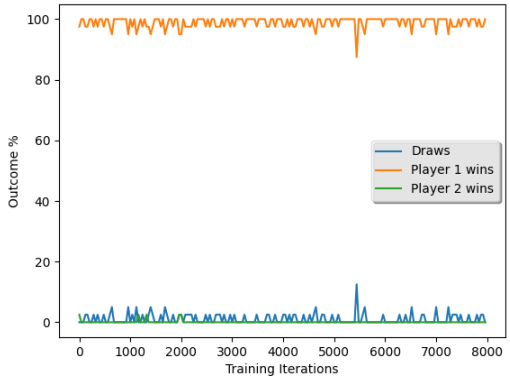


Figure 7: Sarsa vs Sarsa benchmark

Looking at this, the Sarsa agents, much like the Q-learning agent, managed to win almost every game, with non-wins only being draws. This ultimately supports our first hypothesis once again that upon properly training a model, they could play to the caliber in which they never lose a match.

In exploring our third hypothesis, we trained our Sarsa agent against the benchmark player, giving us the

following:

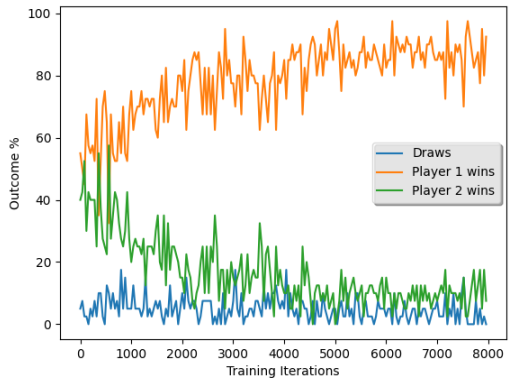


Figure 8: Sarsa training against benchmark.

Similar to our experiment with Q-learning, there is a drastic difference than when the agent trained against itself, as only player 1 (Sarsa) is learning and improving--indicated by the rise in win percentage. As such, evaluating the model by having it not make exploratory moves and playing against the benchmark player, we get:

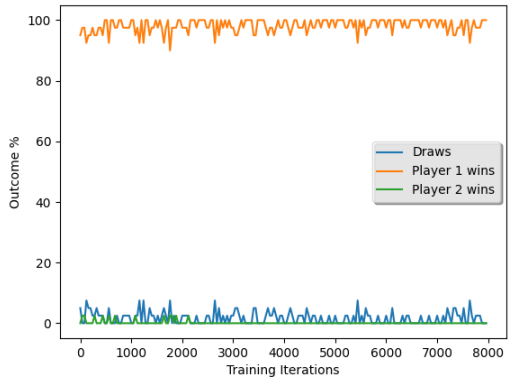


Figure 9: Sarsa vs benchmark benchmark.

Unlike Q-learning, we can see that the Sarsa model, training against the benchmark, managed to still reach a point in which it never lost a game. This indicates both that the number of iterations required and the performance is similar--meaning that our third hypothesis is neither supported or unsupported.

Looking at the Q-learning and Sarsa results, it is difficult to determine whether or not one was substantially faster in training than the other when training against itself. However, when analyzing their results training against the benchmark, we see that the Sarsa agent ended up performing much better, indicating that the Sarsa model may have a slight edge over Q-learning when it comes to required iterations and overall performance. As such, this works to disprove our second hypothesis.

## Conclusion

Overall, in looking at our results, we can first accept our hypothesis that we can design and train an agent to never lose a match of Tic-Tac-Toe. This is especially evident when looking at Figures 2 and 7 in which after training, the Q-learning and Sarsa agent never lost a match.

Next, we need to reject our second hypothesis that Q-learning requires less iterations and is better performing than Sarsa. Though this comparison is somewhat difficult to make, looking at Figures 4 and 9 gives us some indication. These figures show us that despite both models training against the benchmark player, Sarsa was able to reach the point where it never lost while Q-learning was not able to in 8000 training iterations.

Looking at our third hypothesis, we reject it as once again looking at Figures 4 and 9, we see that the Q-learning agent was not able to reach a similar performance than when training against itself, and the Sarsa agent proved to be very much similar to when it trained against itself. Primarily due to the Q-learning agent lacking in performance, we reject the idea that training against the benchmark player is more fruitful.

Lastly, along the same lines, we reject the idea that Q-learning requires less iterations than Sarsa when training against the benchmark player. Again looking at Figures 4 and 9, we see that only Sarsa was able to reach a point during the 8000 iterations in which it would never lose. As such, we can conclude that Sarsa trained better than Q-learning when utilizing the benchmark player.

Ultimately, we were able to achieve our goal of designing a Tic-Tac-Toe agent that could learn to never lose a match through both Q-learning and Sarsa. Furthermore, we were able to explore various nuances of the models through the experiments that we conducted.

### Future Work

In the future, we would like to implement eligibility traces into our learning algorithms, in order to assign credit to moves that lead to a win such that the agent could achieve more wins when it would otherwise draw.

Furthermore, we could further explore the effectiveness of each algorithm by testing them against each other, thereby analyzing which algorithm wins more often. Additionally, we could broaden the scope of our analysis to more reinforcement learning algorithms than those we discussed.

## References


Bobyrev, K. (2018). Reinforcement learning vs Tic Tac Toe: Temporal difference (TD) agent that beats tic tac toe game through self-play. Reinforcement Learning vs Tic Tac Toe. Retrieved March 20, 2022, from https://gist.github.com/kirillbobyrev/a24a811fbd7cd816916c1e04d87efa7a

Schaefer, Steve (2002). "MathRec Solutions (Tic-Tac-Toe)". Mathematical Recreations. Retrieved March 20, 2022.

Sutton, R. S., Bach, F., & Barto, A. G. (2018). Q-learning: Off-policy TD Control. In Reinforcement learning: An introduction (2nd ed., pp. 131–132). essay, MIT Press Ltd.

van de Steeg, M., Drugan, M. M., & Wiering, M. (2015). Temporal Difference Learning for the game tic-tac-toe 3d: Applying structure to Neural Networks. 2015 IEEE Symposium Series on Computational Intelligence. https://doi.org/10.1109/ssci.20