Project III Report for COM S 4/5720 Spring 2025: A Q-Learing Implemenation

Noah Miller[1]

Abstract— **The problem of 3 chasing agents is difficult. These three agents, Tom, Jerry, and Spike, where Spike wants to attack Tom, Jerry wants to attack Spike, and Tom wants to attack Jerry. Not only is this problem being solved, but there is a chance that any action taken may be changed by an unkown probability. This paper details an implementation of Q-Learning as the planning algorithm of choice for these agents, as well as developments made along the way.**

I. INTRODUCTION

It is common for a path seeking problem involving agents to be used when understanding machine learning. Once past the basic understanding of one agent attempting to find the shortest path to a goal, is to introduce a moving goal. One step past that is to include someone to avoid as well. Once all that is met, we are left with the three agent problem. This is a problem in which the three agents, Tom, Jerry, and Spike, attempt to capture each other whilst avoiding obstacles on the map. Each round the map is different, so it needs to allow variance. From these turn, the things to consider are what moves are possible, and that Spike wants to capture Tom, Tom wants to capture Jerry, and Jerry wants to capture Spike. These considerations change the problem from a simplest shortest path algorithm, to one needing more heuristics and understandings of the ”best” path forward. The ultimate goal is to not be captured, but one can easily change the preference of actions based off scoring the winning states. These actions though, may be changed in an unknown probability by the managing algorith. There are 3 changes, keeping the action the same, rotating to the left by 90-degree, and rotating to the right by 90-degree. As such the model must not calculate an algorithm for shortest path, but also one that will be the most likely to succeed given the randomness.

II. ALGORITHMIC DEVELOPMENT

A. Q-Learning

As discussed above, Q-Learning is a non model based learning algorithm to determine best action. The algorithm instead updates a table that is used to determine the best action based upon the state, the position of players, and what player is acting. The algorithm, much like MCTS runs a series of simulations, but this time it is instead entire games simulated. Once these games have been simulated, a lookup table with scores of actions will exist and be used to determine the best action at each stage of the actual game being played, although there is still a chance for the agent to act randomly.

[1]Noah Miller with the Department of Computer Science, Iowa State University, Ames, IA 50013, USA `nvmiller@iastate.edu`

1) Implementation: This implementation again owes its credit to Professor Weng's sample code. This sample code was updated to no longer reflect a game of tic-tac-toe, but rather a game of agents chasing agents, but the template was used as the basis of the logic. The current implementation loops through 100 games, at 100 actions taking place. These games allow for the Q-table to be updated according to this scoring. 3 points if the acting player captured the pursued, -3 if they have been captured, 1 if the other agents have captured each other, and finally 1 point if the number of rounds has been exceeded. These points (reward) is used in the following way.

$$Q(s,a,p) = Q(s,a,p) + \alpha(\text{reward} + \gamma q - Q(s,a,p))$$

s is the state, a is the action, p is the player taking the action, α is the learning constant, 0.5, γ is the discount factor, 0.9, q is the max next state score, and $Q(s,a,p)$ is the score of the current action. A state is this game is the position of each of the players, and an action is a movement 1 square in any direction, or standing still. This algorithm is once again naively implemented which will be discussed later, but that is how the Q-table gets updated. The first change from the tic-tac-toe implementation to the current was changing what actions can take place. Once these actions were understood, manipulating the other methods to be able to feed the information was important. Most methods needed to get updated to include the player acting as it is no longer just 2 and can be checked simply, but rather needed to be tracked throughout the entire process. The learning does still have a 10% chance to act randomly at any given point, allowing for training of non-optimal positions, as well as not always making non-optimal moves.

2) Testing: There was a very small amount of testing done in order to determine what iteration amounts to use. The solution started much higher, at 1000 games at 500 actions taking place, but this amount of computation was leading to results for the games taking far too long. As such the iteration amounts were slowly decreased first the number of games was reduced to 500 as well, and then down to 200, and then lowering the action amount to 300, and finally where the solution currenlty stands at 100 games with 100 actions taking place. More testing could have been done to determine what learning constants, discount factors, randomness, and scoring algorithms allow for the most optimal model.

III. DISCUSSION

While this solution has the most promise of determining good actions to take at any point in the board, there are still some inefficenieces and places for improvement if work were to continue towards solving this problem. One heuristic not included in training the model is how other players will act. Currently other players react under the same model, but this will not be the case after submission, where it will instead be facing other models. As such the moves it believes to be optimal, may not be optimal or taken by other models leading to a diverging solution path very quickly. That is the reason for the ability to randomly sample, but is something of note. Another problem is that this model did not get as much testing as is avaible for it. The learning, discovery, and random sampling constants were simply used from Professor Weng's sample code. As such these constants may not be refined for the problem at hand. Another inefficenciey is the speed. The speed is faster than MCTS, but because of the depth of the problem, there are still a lot of actions to check and calculate scores upon. The depth of the problem is the biggest issue, as there is no way to explore all possible options. As such this method explores those that are most likely to be explored by others. The depth problem also only occurs once at the start as opposed to potentially happening on every action as in MCTS

IV. CONCLUSIONS

Based off the limited testing and naive attempts at implementation of Q-learning and MCTS, Q-learning is more efficient on a time scale capacity without large noticeable loss in the algorithmic succcess. This claim is made very loosely though as sufficient testing has not been done. Q-Learning allows for the quickest responses and has been tested the most for depth to test, and how many loops to test upon. Q-Learning also seems the most adoptable towards the 3 agent problem from the simpilier 2 agent implementation detailed in Professor Weng's sample code.