Carlos Lazo
CSCI-4150
Homework #6
Q-Learning Write-Up

It is completely realistic to formulate a Q-learning algorithm to develop an intelligent Tic-Tac-Toe agent over time. As far as developing the agent's intelligence when making a move, I would use the following pseudo code in the listed priority order for it to help determine what moves it should make:

AI Agent Get-Move Pseudo Code

> *If – There exists a board position x which is empty where agent can win by placing its piece there → Return x (offensive move) and update previous state's Q value with reward for winning.*
>
> *Else if – There exists a board position x which is empty where the opponent can win by placing its piece there → Return y (defensive move).*
>
> *Else –*
>
> > *If – Probability factor of exploration is evoked (using random # generator, which will be explained later) → Return random open position*
> >
> > *Else → Choose move which has maximum Q value of current state s.*

In order to train the agent, a piece of code/software could be written which pits the AI against a computer player which chooses moves at random. This would help ensure that all possible game states have the potential of being explored, independent from the decisions of our AI agent. The software will be able to tell when either the agent or the random computer has won or lost. Also built into the algorithm will be a variable to determine exploration vs. exploitation. Initially the software will initialize the variable to have a high probability where the agent will choose random moves to learn the behavior of the game and its respective states. As more and more games are played, the randomness factor will be decreased and the movement decisions made by the agent will become more dependant upon the Q values learned throughout the process. As far as

a plan to use the software, it will be run enough times for the Q values to become refined and for clear and specific policies to be developed and seen by the AI agent.

Utilizing a list of nine elements to keep track of the each of the nine possible tic-tac-toe slots, along with a small database to keep track of Q values spanning from one state to another, they can be updated fairly intelligently. All Q values in the database will be initially set to 0, with a gamma discount factor of .9 to help the accuracy of the program. If the agent encounters a state where it can pick a move that leads to a win, then the Q value between the current state and the goal state will be set to a reward of 100. If the agent, in retrospect, encounters a state where it can pick a move that leads to a loss, then the Q value between the current state and the loss state will set to a reward of -100. In other words, bad moves will be penalized in terms of the AI agent. Moves that lead to a draw or that yield no win or loss will have the Q value set to the max Q value of the next state plus a reward of 0.

The randomness of the both the initial agent (with a simultaneous decreasing variable probability and intelligence increase) and the random opponent training program will help ensure that a vast range of moves are tried during the learning process. The implementation of penalizing Q values when loss states are found will help the AI agent pick more intelligent moves, therefore minimizing the chances of it losing. It will only choose a loss state if it absolutely has to in this sort of implementation. A realistic estimate, after the software is run numerous times, would ensure that the intelligence of the agent in respect to Q values would converge on a fairly accurate and consistent winning policy.

Note: One addition that would help make the AI agent more accurately could be to train it against another learner agent – or against itself. Playing an optimal player vs. a random player would further help increase the accuracy of Q values.