Daniel Fernández
932-063-474
ME538
20 October 2014
Homework #1

The assignment calls for the construction of a 5 x 10 gridworld occupied by a system of agents and targets moving to adjacent cells on an iterative basis. The user is to implement a learning algorithm which will aid the agent(s) in "capturing" the target, where a capture is the simultaneous occupation by target and agent of the same cell. The report is divided into three parts. In Part 1, the target will be not be given any specific path, and will travel randomly. In Part 2, the target will be allowed to move away from the agent if that action is possible, and will operate randomly if not. In Part 3, two agents will be introduced to illustrate any possible benefit of a multi-agent system. In all 3 Parts, the overarching algorithm to be applied to the system is the Q-Learning Method:

$$Q_{t+1}(s_t, a_t) = \underbrace{Q_t(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha_t(s_t, a_t)}_{\text{learning rate}} \times \left[\overbrace{\underbrace{R_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \underbrace{\max_a Q_t(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q_t(s_t, a_t)}_{\text{old value}} \right]$$

[from wikipedia.org]

In the above algorithm, the ideal future state is computed by using previous data at that state, any possible rewards from future states, and some additional weights factored to them. In application, a new Q value is obtained by taking its previous value and adding a "learned" value to it. After sufficient iterations, this Q-value will give the highest probability of successfully finding the target with the least amount of timesteps.

Part 1 – Target Random Motion Learning

In Part 1, the target will move randomly across the map, with the agent iterating through the qlearn.m function. As inputs, this function receives the agent and target locations, the world map, Q and R matrices, and a counter variable. It outputs a new agent position as well as an updated world map. The agent has the advantage of allowing the target to move first in a chess-like manner, knowing the target's state prior to moving itself. Through each iteration, the Q-matrix is updated and applied to successive moves. Of note, there is an epsilon value of 0.1 which gives the agent a 10% chance to ignore the Q-Learn array output and move randomly. This is to promote active exploration of the world map, and help from any previous data "dominating" the agent's movement.

Part 2 – Target Active Evading Motion Learning

In Part 2, the target is allowed a simple reckoning algorithm to choose an evasive pattern. By using an absolute function, the targetsmart1.m function moves the target away from the agent as possible. If there is no optimal step, the target moves randomly. This delays the agent from finding the target as compared to Part 1. It is capable of capturing the target, usually by way of a "cornering" effect, where the absolute value method breaks down. Admittedly, the author could have implemented a more robust positioning algorithm to make the target more elusive.

Part 3 – Target Active Evading Motion Learning with 2 Agents

In Part 3, the system is updated to include 2 agents, making it a legitimate albeit simple Multi-Agent (MA) System. The targetsmart2.m function is introduced to allow the target to compensate for the additional agent. The qlearn.m function is not updated; however, allowing the second agent to use the same Q-Matrix to take its action. This may have not been the instructor's intention as it demonstrates MA cooperation. The second agent has a slight advantage, being able to move after the target and agent1 during each iteration. This method proved the most effective, as the target tended to move randomly more often than not, rather than actively evade the agents.

Appendix – MatLab Script

[see attached files for MatLab scripts]