CSCI 446 Artificial Intelligence Project 4 Design Report

Roy Smart Nevin Leh Brian Marsh

December 2, 2016

1 Introduction

2 The Racetrack Problem

3 Reinforcement Learning Algorithms

4 Value Iteration

4.1 Value Iteration

Value iteration is a sequential reinforcement learning method for determining the optimal policy for a Markov decision process(MDP)[Yu and Bertsekas, 2013]. The algorithm calculates the utility for all of the states and then uses these utility values to determine the optimal action a for each state s. To calculate utility we will be using the *Bellman equation* as described in [Russell and Norvig, 2010]. The equation is given as

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s') \tag{1}$$

where R(s) is the reward for the current state, s' is the next state, and $P(s'|s,a)$ is the probability of being in state s' given an action a in state s. γ is used to tweak how much weight we put on the expected next states utility values for calculating the current states utility value.

Value iteration uses this equation to iteratively propagate utilities and actions to each state. To start each state is given a random utility, though each state still has a unique reward value with the goal states having high reward. We then iterate over every state, recalculating each states utility. We keep looping through the states until two full loops through the states do not result in a change of utility for any of the the states. We can then assign optimal actions for each state depending on that states utility.

4.2 Design Implications

We will need to have some type of data structure for keeping the representation of our track, with space for each states reward values, utility values, and the optimal action. We can either use a large dimension array to hold these values or several three dimensional arrays for each value. In addition we should make γ tunable to see how that alters training time and results. We will also have to implement a way to handle walls and make sure that moves through a wall cannot happen.

5 Q-Learning

5.1 Description

Q-learning is a model-free reinforcement learning method for determining optimal action-selection policies [Russell and Norvig, 2010]. An agent using this method leverages a quantity known as the Q-value to derive the optimal action a for each state s. The Q-value, $Q(s, a)$ describes the expected utility for every action in every state within the environment and is learned by the agent using temporal difference learning. An expression to calculate the Q-value is given by [Russell and Norvig, 2010] as

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \tag{2}$$

where a is the action that was executed in state s that resulted in state s' and $R(s)$ is the reward function. The constants α and γ are known respectively as the learning rate and the discount factor. Equation 2 is used as an update rule to adjust the value of $Q(s, a)$ for each action-state pair in every time trial undertaken by the agent. Using this simple update rule and randomly initialized Q-values for every action-state, an agent can learn how to navigate an environment.

5.2 Design Implications

Since the racetracks are reasonably small, we can afford to sore the table of Q-values in memory. We will base our Q-learning agent off of the function Q-Learning-Agent provided by [Russell and Norvig, 2010]. To ease the training time, we will incrementally train the Q-learning agent by beginning training close to the finish line, and then increasing the distance as the agent learns each section.

6 Software Design

For this project we will be using an environment and agent model similar to the wumpus world. Instead of a board we will have a `Track` class that reads in the different tracks from a text file. To run the experiments we will have an `Environment Engine` for each `Track` and `Agent` combo.

The `Agent` will be a virtual class that contains data such as the max acceleration values. It will also contain the virtual method `learn`. Each concrete class will be the implementation of our reinforcement learners. Each of these classes will contain an overridden learn method and algorithm specific functionality. The key difference between the wumpus world `Agent` and this `Agent` is the fact that in this case the `Agent` can see the whole board from the start.

7 Experiment Design

For each agent we will generate learning curves that describe the number of steps required to reach the goal state vs. the number of training iterations. We will also track the number of times each agent runs into a wall and needs to be restarted. Each agent will be trained at least 10 times to determine reliable learning curves.

8 Summary

References

[Russell and Norvig, 2010] Russell, S. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Pearson Education, Upper Saddle River, New Jersey 07458, 3rd edition.

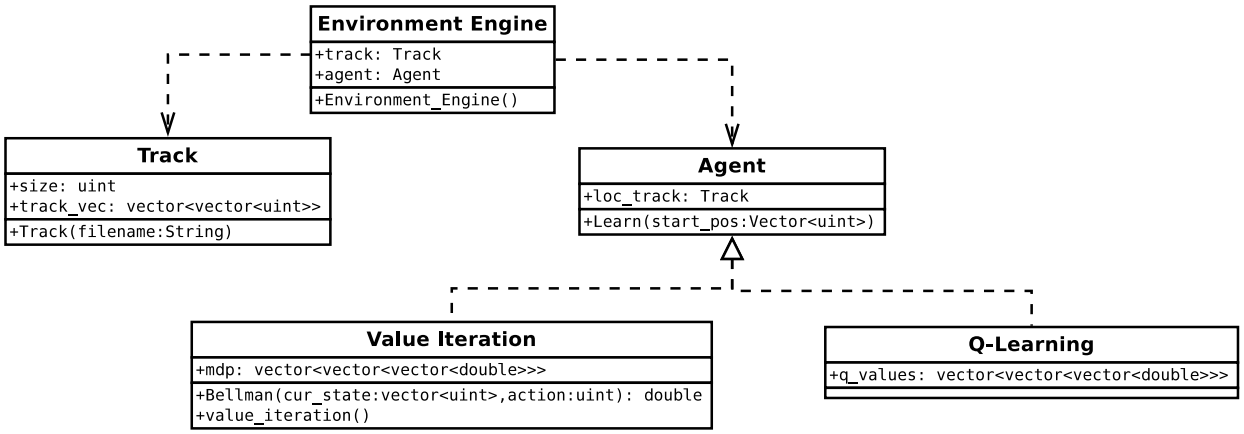

Figure 1: *UML of our design*

[Yu and Bertsekas, 2013] Yu, H. and Bertsekas, D. (2013). Q-learning and policy iteratin algorithms for stochastic shortest path problems. *Annals of Operatons Research*, 208(1):95–132.