COMPSYS 725: Reinforcement Learning + Planning with POMDPs

Andrew Lai (klai054)
Department of Electrical and Computer Engineering

27th October 2017

1 Introduction

This report will outline the creation of the models and a discussion of any decisions made on each section done within the assignment. Two underlying problems separated as a study of Reinforcement Learning and a POMDP, both scenario of modelling with robots navigating within a specific predefined domain.

The Reinforcement Learning problem was solved with Java and the POMDP was solved with an existing solver called the Approximate POMDP Planning Software[1].

2 Reinforcement Learning

Refer to the `README.md` file as to know how to run the RL program. The RL problem is sovled with Java, this because prior experience with the Java language to create simplistic GUIs to better visualise the RL algorithm in action.

Fig. 1 illustrates the domain the RL brain is working in. The yellow circle represents the robot location. The black squares in the grid represents danger zone and the green zone being the goal state. The numbers in each grid represents the reward the world emits once entering the zone. Moving in to a danger zone is negative rewards of a large magnitude of -100 as entering danger zones should never happen. In contrast the goal state has a large reward value of 100 to entice the learning to enter this goal state.

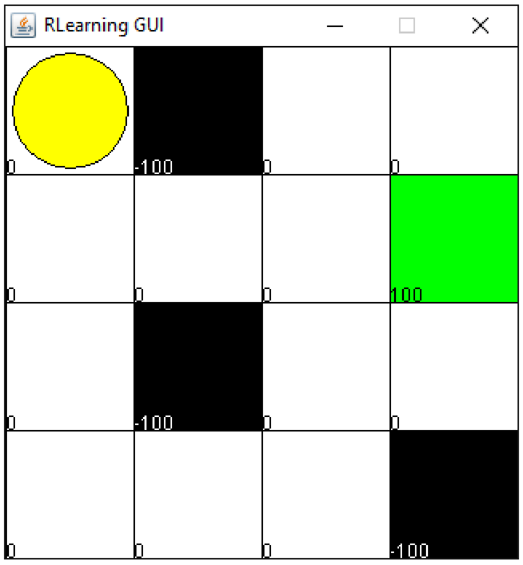

Figure 1: 4 X 4 grid world of the RL problem. Numbers illustrating reward values

[1]http://bigbird.comp.nus.edu.sg/pmwiki/farm/appl/

Q-Learning is the RL model-free method implemented for obtaining the optimal policy for the given domain. The program is developed in three separate class files as follows:

- `RLBrain`: This is the brain class containing the Q table and eventually containing the optimal policy. This brain has no way of knowing where danger states are at and is only capable to the observe current state.
- `RLWorld`: The world contains the grid of the domain/world, where this holds the locations of the robot's current position, danger positions and goal positions. This is what gives the robot the observations of its current state and also gives the rewards (penalty) for the robots actions. The main functionality of the visualisation program is declared here, such as the GUI elements.
- `RLearning`: This is where the main function is contained, its task is also to join the world and the brain together. This is also where the value iteration is controlled.

When a full set of episodes are complete the GUI will display the policy in place of the rewards prior to the learning execution, as seen in Fig. 2.

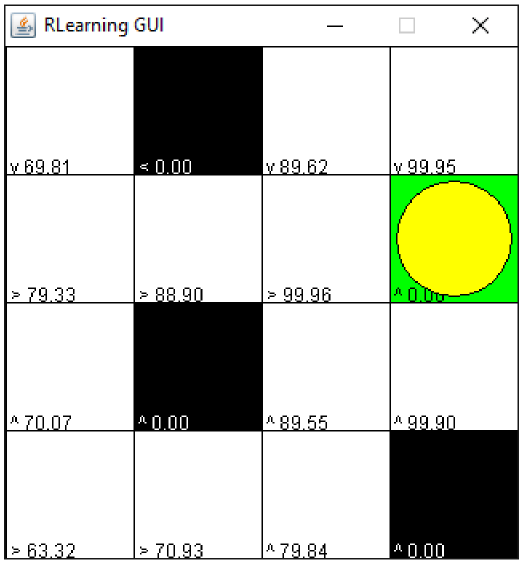

Figure 2: GUI showing learnt policy

The prefix can be of either these four characters: $\wedge, \vee, <$ and, $>$ each representing the direction up, down, left and, right. The number printed next to it is the value for taking said action. Figure 2 also showcases the situation after one thousand episodes, this image can be found in the `../policies` folder given in the deliverable along with other images showing different number of episodes. This is done to show that, given there are more samples, more experience is gained and therefore the brain would be able to gauge a better policy.

The episodes are made to terminate when entering the goal state to maximise policies to enter this state and not move out. This is also done for entering unsafe zones. Once an episode terminates, the robot is reset to another random position that is not in either the goal or the danger zones.

The Q-learning updates values with the equation 1:

$$value = Q + \alpha(r + \gamma maxQ - Q); \tag{1}$$

In the case of extra credits, another learning RL algorithm is implemented for the same problem. The algorithm is namely SARSA, this is toggled with the Boolean variable found in the `RLearning` *file where values are computed differently with equation 2.*

$$value = Q + \alpha(r + \gamma nextQ - Q); \tag{2}$$

While working on the RL section of the assignment, I was able to be actively engaged in the research of RL algorithms. I have come to realise how powerful these concepts are.

3 Planning with POMDP

The Approximate POMDP Planning Software tool was used to model and solve POMDP problem laid out.

The `.pomdp` file contains all the definitions of the transition, observations and rewards. The following are the states found in the POMDP problem.

- `door0_t..door4_t`: these are states which defines which single door the treasure is behind.
- `terminated`: once an action is taken to commit to a specific door the state is set to terminated.

The actions are split into the following:

- `open0..open4`: these set of actions is used to get observations of the presence of treasure.
- `take0..take4`: these set of actions is used to commit to the specific door that the treasure may be behind.

Observations can either be: if the treasure is seen or not. These are in correspondence with the fact that if a door is checked, the reward is set as penalties/negative reward values based on the effort that is required to scan a room. If a take command is done, and the committed answer is correct, then a large positive reward is given. No reward effect is given for taking a room if the room does not have the treasure.

I believe that the way the model that has been implemented could have the reward values changed a bit to make the POMDP better in the fact that if the robot were to the take incorrect actions, heavy punishment should be set. However, adjustments of these reward values changes how the values converge, which I had observed while working with the POMDP solver. While working on this assignment I have furthered my understanding of POMDP. The use of the POMDP is a strong tool to allow for robots for to learn, once the model is set (like as the problem modelled in this assignment), tuning the reward values and optimisation becomes the challenge.