# MSALT7 Reinforcement Learning: Coursework

Milica Gašić, Carl Edward Rasmussen

Due: March 3, 2016

In this assignment, you will implement the basic building-blocks of Reinforcement Learning for a discrete grid-world model. We will provide code (located at `http://mlg.eng.cam.ac.uk/teaching/mlsalt7/1516/code.zip`) for constructing the models and plotting their output in `matlab`.

a) 24% : Implement the value iteration algorithm. This should be implemented as a function

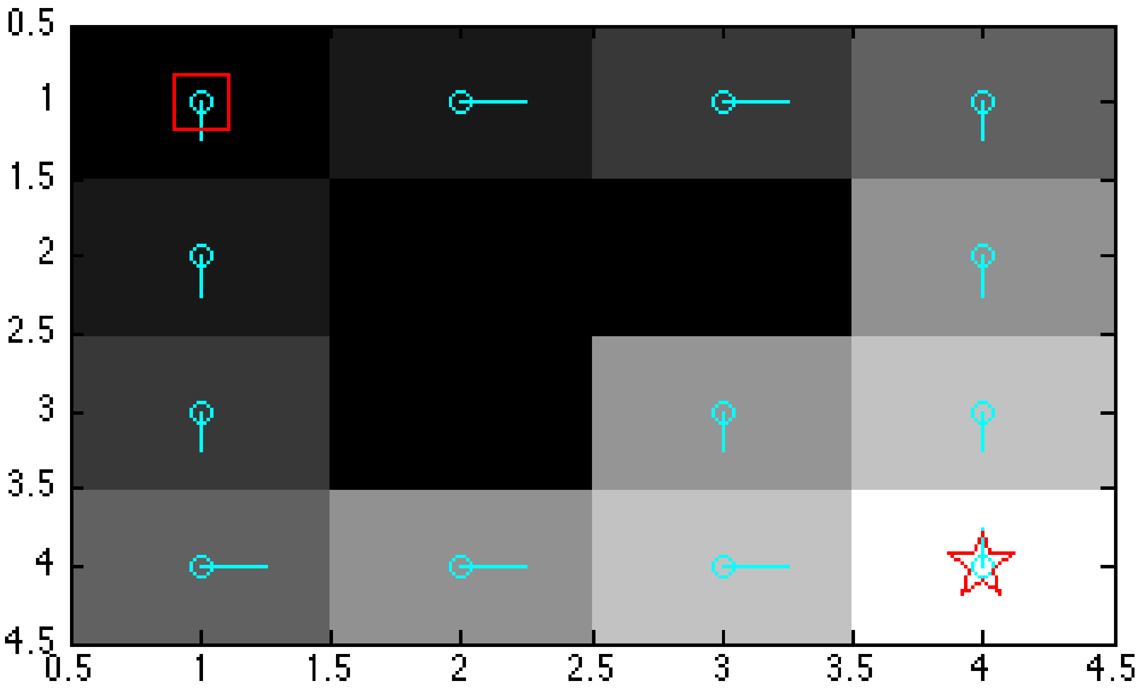
```
[v, pi] = valueIteration(model, maxit)
```

which, given a model of the environment and a maximum number of iterations, returns the optimal value function as an array of floats and the optimal policy as an array of integers. We have provided 3 scripts which will create the model: `smallworld`, `gridworld`, and `cliffworld`. For example, the following code can be used to compute and plot the value function and policy:

```
smallworld;
[v, pi] = valueIteration(model, 100);
plotVP(v, pi, paramSet);                % paramSet is also created by
                                        % the smallworld script
```

The `model` parameter should be a structure which contains the number of states in `model.stateCount` as well as arrays `model.R(s, a)` and `model.P(s, s_, a)` which represent the reward for taking action `a` from state `s` and then the probability for transitioning to state `s_`. Note: you should also implement an early stopping rule or convergence check for value iteration

Turn in your code for value iteration and a plot of the value function and policy for the model created by the `gridworld` script. For testing purposes we have included the value function for `smallworld`, which should look like:



b) 24% : Implement the policy iteration algorithm as a function `[v, pi] = policyIteration(model, maxit)`. Turn in the code for this algorithm and a plot for the optimal value function and policy for the `gridworld` model.

c) 19% : Implement the SARSA algorithm as a function `[v, pi] = sarsa(model, maxit, maxeps)` which takes a model, a maximum number of iterations per episode, and a maximum number of episodes. Here an episode is the trajectory or sequence of steps that starts from the initial state and ends in the goal state. By limiting `maxit` we allow the algorithm to "start over" if it is taking too long to find the goal state, however do not make this value too small or the algorithm will never be able to learn. Note that `model.startState` and `model.goalState` are defined so that this can be checked during the algorithm's progress. Turn in your code and a plot of the value function for `smallworld`.

d) 19% : Implement the Q-learning algorithm as a function `[v, pi] = qLearning(model, maxit, maxeps)` which takes a model, a maximum number of iterations per episode, and a maximum number of episodes. Turn in your code and a plot of the value function for `smallworld`.

e) 14% : Modify the SARSA and Q-learning algorithms to save the accumulated reward per episode. Run these algorithms using the `cliffworld` model and plot the rewards obtained throughout the learning process. This should look similar to Figure 6.13 from the Sutton and Barto text (`https://webdocs.cs.ualberta.ca/~sutton/book/ebook/node65.html`). Your plot may look slightly different (and likely less smooth) due to the values you choose for $\epsilon$ and $\alpha$.