# ASEN 5519-003 Decision Making under Uncertainty
# Homework 3: Online MDP Methods

February 11, 2023

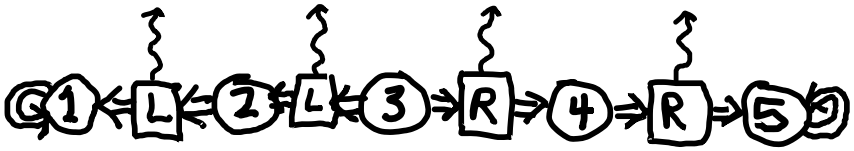
## 1 Conceptual Questions

**Question 1.** (20 pts) Do similar $Q$ values imply similar rewards?

In the proof for Lemma 5 of the Sparse Sampling paper by Kearns, Mansour, and Ng,[1], the authors make the following claim:

> If a policy $\pi$ satisfies $|Q^*(s, \pi^*(s)) - Q^*(s, \pi(s))| \leq \beta$ for all $s \in \mathcal{S}$, then it immediately follows that $|\mathcal{R}(s, \pi^*(s)) - \mathcal{R}(s, \pi(s))| \leq \beta$.

In this exercise, you will demonstrate that this claim is mistaken. Consider the MDP below:




The state space is $S = \{1, \ldots, 5\}$ and the action space is $A = \{L, R\}$ (but not all actions are available from each state). Transitions are deterministic as shown. The discount factor is $\gamma = 0.9$.

Choose a reward function, $\mathcal{R}$, (i.e. values for the squiggly arrows), a policy, $\pi$, and a value $\beta$ that constitute a counterexample to the claim above. Justify your answer.

## 2 Exercises

`HW3.DenseGridWorld()` generates a 60x60 grid world MDP. There is a reward of +100 every 20 cells, i.e. at [20,20], [20,40], [40,20], etc. Once the agent reaches one of these reward cells, or an edge cell, the problem terminates. All cells also have a cost. Only a generative transition model is available. You will use the following functions from POMDPs.jl to interact with this problem (or larger versions) in the rest of this assignment:

- `actions(m)`
- `@gen(:sp, :r)(m, s, a)`
- `isterminal(m, s)`
- `discount(m)`
- `statetype(m)`
- `actiontype(m)`

[1] `https://www.cis.upenn.edu/~mkearns/papers/sparsesampling-journal.pdf`; Note: you do not need to read the paper to complete the problem.

**Question 2.** (15 pts) Monte Carlo Policy Evaluation

a) Write a rollout simulation function for an MDP starting with the following code:

```
r_total = 0.0
t = 0
while !isterminal(mdp, s) && t < max_steps
    a = :down # replace this with a policy
    s, r = @gen(:sp,:r)(mdp, s, a)
    r_total += discount(m)^t*r
    t += 1
end
```

Use this function to perform a Monte Carlo evaluation of a uniform random policy on an MDP created with `HW3.DenseGridWorld(seed=3)`. Report the mean discounted reward estimate and standard error of the mean (SEM). Run enough simulations so that the SEM is less than 5.

b) Create a heuristic policy that improves upon the random policy by at least 50 reward units. Report the mean and standard error from a Monte Carlo evaluation.

**Question 3.** (20 pts) Monte Carlo Tree Search

Write code that performs 7 iterations of Monte Carlo Tree Search on an MDP created with `HW3.DenseGridWorld(seed=4)`, starting at state $(19, 19)$. You will need to produce three dictionaries:

- `Q` maps $(s, a)$ tuples to Q value estimates.
- `N` maps $(s, a)$ tuples to N, the number of times the node has been tried.
- `t` maps $(s, a, s')$ tuples to the number of times that transition was generated during construction of the tree.

Then visualize the resulting tree with `HW3.visualize_tree(Q, N, t, SA[19, 19])`[2]. **Submit an image of the tree, the code used to generate it, and a few sentences describing the tree after 7 iterations** (e.g. which actions have the highest Q values? Does this make sense?).

**Question 4.** (15 pts) Planning with MCTS

Use your Monte Carlo tree search from Question 3 to plan online in the simulation loop. Limit the planning time to 50ms. Evaluate the MCTS planner with 100 100-step Monte Carlo simulations. Report the mean accumulated reward and standard error of the mean, along with a typical number of iterations that MCTS was able to complete each time it chose an action[3].

# 3 Challenge Problem

**Question 5.** (10 pts code and description, 20 pts score) Fast Online Planning

Create a function `select_action(m,s)` that takes in a $100 \times 100$ `DenseGridWorld`, `m`, and a state `s`, and returns a near-optimal action within 50ms. You may wish to base this code on the MCTS code that you wrote for Question 3. Evaluate this function with `HW3.evaluate` and **submit the resulting `json` file along with the code and a one paragraph to one page description of your approach**, including tuning parameters that worked well, the rollout policy, etc. A score of 50 will receive full credit. There are no restrictions on this problem - you may wish to use a different algorithm, multithreading, etc. Starter code on github will give suggestions for timing and other details.

[2] `SA` is from the `StaticArrays.jl` package.

[3] You can determine a typical number of iterations by just printing out the number; you don't need to keep careful statistics unless you want to