Markov Decision Process

Recycling Robot Example

Recall the recycling robot example that we covered in the Reinforcement Learning lesson. The robot's goal was to drive around its environment and pick up as many cans as possible. It had a set of **states** that it could be in, and a set of **actions** that it could take. The robot would receive a **reward** for picking up cans, however, it would also receive a negative reward (a penalty) if it were to run out of battery and get stranded.

The robot had a non-deterministic **transition model** (sometimes called the *one-step dynamics*). This means that an action cannot guarantee to lead a robot from one state to another state. Instead, there is a probability associated with resulting in each state.

Say at an arbitrary time step t, the state of the robot's battery is high ($S_t = high$). In response, the agent decides to search for cans ($A_t = search$). In such a case, there is a 70% chance of the robot's battery charge remaining high and a 30% chance that it will drop to low.

Let's revisit the definition of an MDP before moving forward.

MDP Definition

A Markov Decision Process is defined by:

- A set of states: $\mathcal{S}$,
- Initial state: s_0,
- A set of actions: $\mathcal{A}$,
- The transition model: $T(s, a, s')$,
- A set of rewards: $\mathcal{R}$.

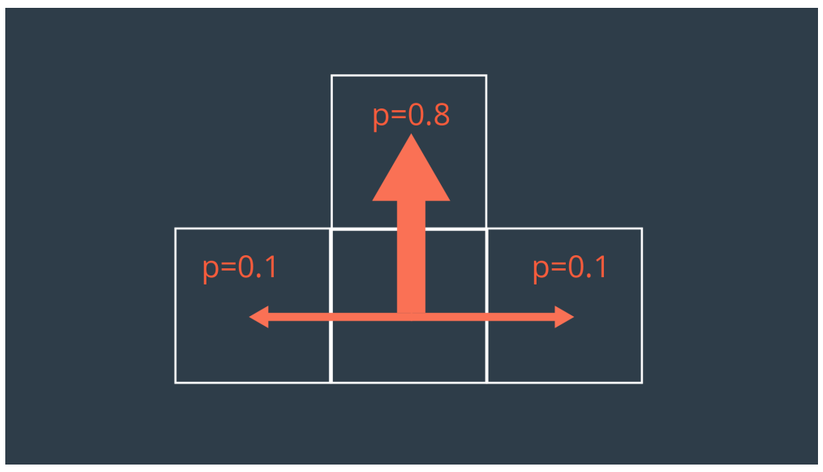
The transition model is the probability of reaching a state s' from a state s by executing action a. It is often written as $T(s, a, s')$.

The Markov assumption states that the probability of transitioning from s to s' is only dependent on the present state, s, and not on the path taken to get to s.

One notable difference between MDPs in probabilistic path planning and MDPs in reinforcement learning, is that in path planning the robot is fully aware of all of the items listed above (state, actions, transition model, rewards). Whereas in RL, the robot was aware of its state and what actions it had available, but it was not aware of the rewards or the transition model.

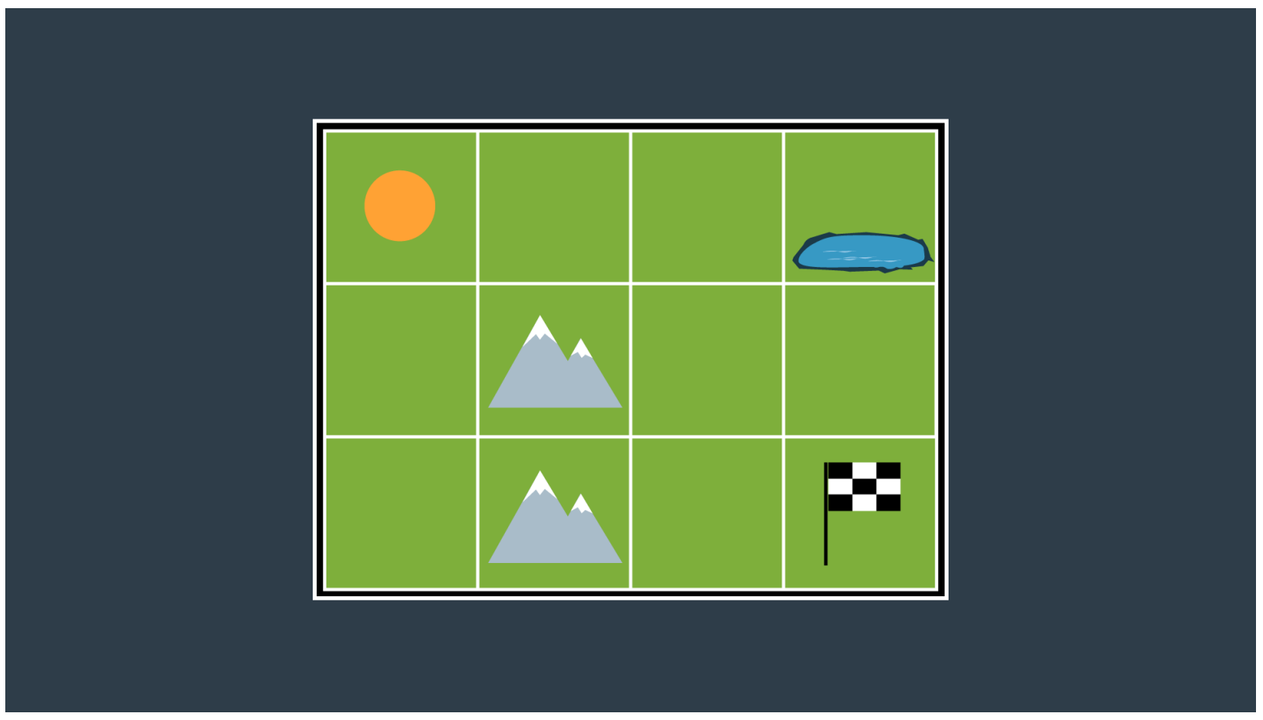
Mobile Robot Example

In our mobile robot example, movement actions are non-deterministic. Every action will have a probability less than 1 of being successfully executed. This can be due to a number of reasons such as wheel slip, internal errors, difficult terrain, etc. The image below showcases a possible transition model for our exploratory rover, for a scenario where it is trying to move forward one cell.



As you can see, the intended action of moving forward one cell is only executed with a probability of 0.8 (80%). With a probability of 0.1 (10%), the rover will move left, or right. Let's also say that bumping into a wall will cause the robot to remain in its present cell.

Let's provide the rover with a simple example of an environment for it to plan a path in. The environment shown below has the robot starting in the top left cell, and the robot's goal is in the bottom right cell. The mountains represent terrain that is more difficult to pass, while the pond is a hazard to the robot. Moving across the mountains will take the rover longer than moving on flat land, and moving into the pond may drown and short circuit the robot.



Combinatorial Path Planning Solution

If we were to apply A* search to this discretized 4-connected environment, the resultant path would have the robot move right 2 cells, then down 2 cells, and right once more to reach the goal (or R-R-D-R-D, which is an equally optimal path). This truly is the shortest path, however, it takes the robot right by a very dangerous area (the pond). There is a significant chance that the robot will end up in the pond, failing its mission.

If we are to path plan using MDPs, we might be able to get a better result!

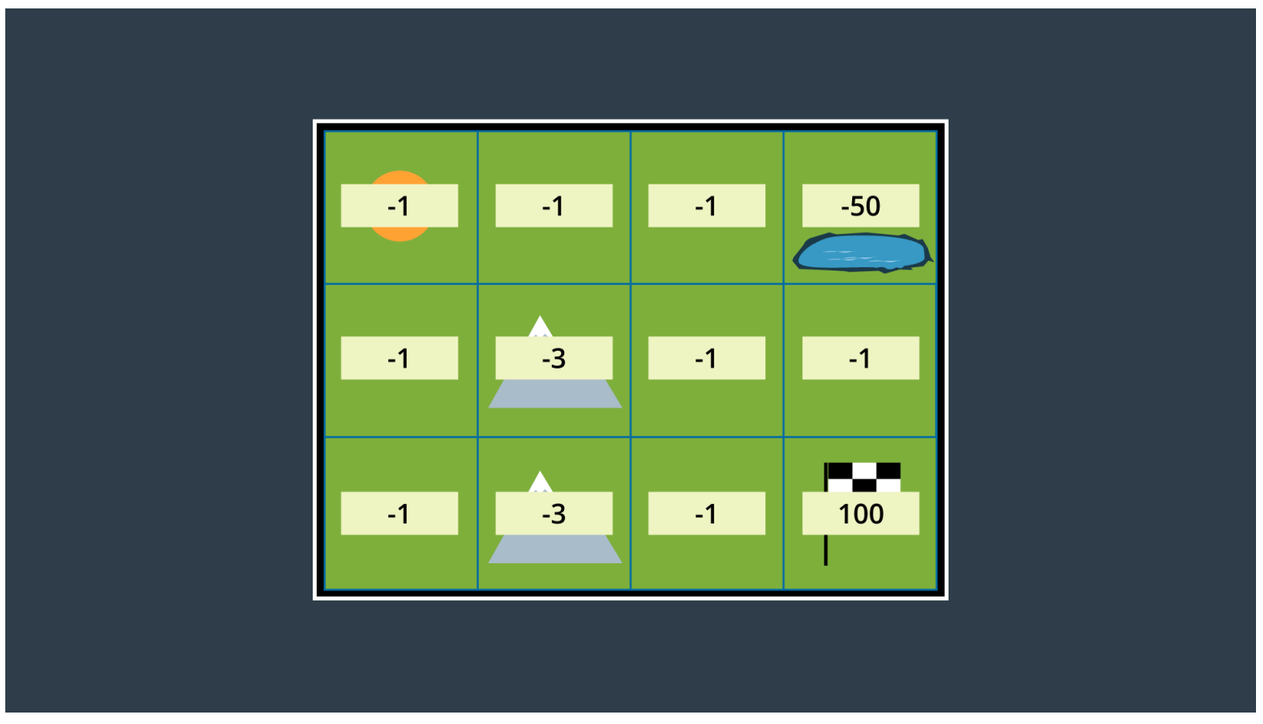
Probabilistic Path Planning Solution

In each state (cell), the robot will receive a certain reward, $R(s)$. This reward could be positive or negative, but it cannot be infinite. It is common to provide the following rewards,

- small negative rewards to states that are not the goal state(s) - to represent the cost of time passing (a slow moving robot would incur a greater penalty than a speedy robot),
- large positive rewards for the goal state(s), and
- large negative rewards for hazardous states - in hopes of convincing the robot to avoid them.

These rewards will help guide the rover to a path that is efficient, but also safe - taking into account the uncertainty of the rover's motion.

The image below displays the environment with appropriate rewards assigned.



As you can see, entering a state that is not the goal state has a reward of -1 if it is a flat-land tile, and -3 if it is a mountainous tile. The hazardous pond has a reward of -50, and the goal has a reward of 100.

With the robot's transition model identified and appropriate rewards assigned to all areas of the environment, we can now construct a policy. Read on to see how that's done in probabilistic path planning!

NEXT

Policies

Recall from the Reinforcement Learning lesson that a solution to a Markov Decision Process is called a policy, and is denoted with the letter π.

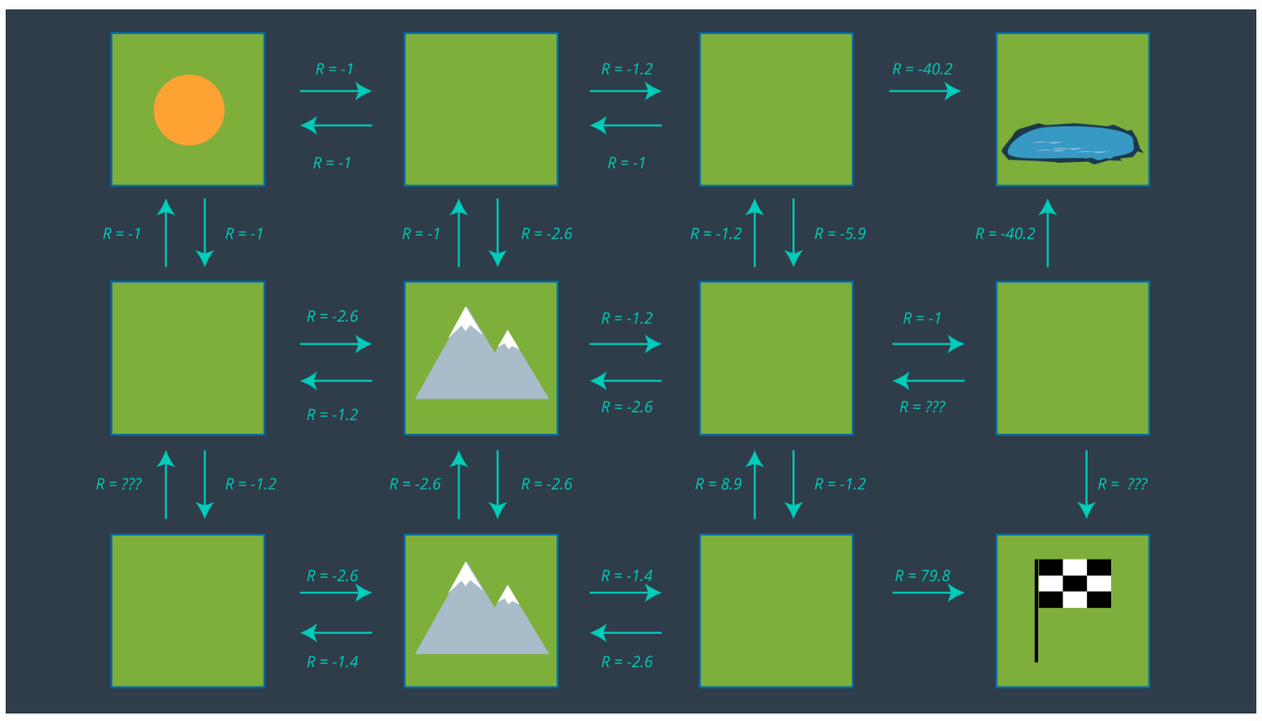
Definition

A **policy** is a mapping from states to actions. For every state, a policy will inform the robot of which action it should take. An **optimal policy**, denoted π^*, informs the robot of the *best* action to take from any state, to maximize the overall reward. We'll study optimal policies in more detail below.

If you aren't comfortable with policies, it is highly recommended that you return to the RL lesson and re-visit the sections that take you through the Gridworld Example, State-Value Functions, and Bellman Equations. These lessons demonstrate what a policy is, how state-value is calculated, and how the Bellman equations can be used to compute the optimal policy. These lessons also step you through a gridworld example that is *simpler* than the one you will be working with here, so it is wise to get acquainted with the RL example first.

Developing a Policy

The image below displays the set of actions that the robot can take in its environment. Note that there are no arrows leading away from the pond, as the robot is considered DOA (dead on arrival) after entering the pond. As well, no arrows leave the goal as the path planning problem is complete once the robot reaches the goal - after all, this is an *episodic task*.



From this set of actions, a policy can be generated by selecting one action per state. Before we revisit the process of selecting the appropriate action for each policy, let's look at how some of the values above were calculated. After all, -5.9 seems like quite an odd number!

Calculating Expected Rewards

Recall that the reward for entering an empty cell is -1, a mountainous cell -3, the pond -50, and the goal +100. These are the rewards defined according to the environment. However, if our robot wanted to move from one cell to another, it it not guaranteed to succeed. Therefore, we must calculate the **expected reward**, which takes into account not just the rewards set by the environment, but the robot's transition model too.

Let's look at the bottom mountain cell first. From here, it is intuitively obvious that moving right is the best action to take, so let's calculate that one. If the robot's movements were deterministic, the cost of this movement would be trivial (moving to an open cell has a reward of -1). However, since our movements are non-deterministic, we need to evaluate the *expected* reward of this movement. The robot has a probability of 0.8 of successfully moving to the open cell, a probability of 0.1 of moving to the cell above, and a probability of 0.1 of bumping into the wall and remaining in its present cell.

$$expected\ reward = 0.8 * (-1) + 0.1 * (-3) + 0.1 * (-3)$$

$$expected\ reward = -1.4$$

All of the expected rewards are calculated in this way, taking into account the transition model for this particular robot.

You may have noticed that a few expected rewards are missing in the image above. Can you calculate their values?

Expected Reward Quiz

What is the expected reward for moving from the bottom-left cell to the cell above it?

-1.2

RESET

Let's look at the bottom mountain cell first. From here, it is intuitively obvious that moving right is the best action to take, so let's calculate that one. If the robot's movements were deterministic, the cost of this movement would be trivial (moving to an open cell has a reward of -1). However, since our movements are non-deterministic, we need to evaluate the *expected* reward of this movement. The robot has a probability of 0.8 of successfully moving to the open cell, a probability of 0.1 of moving to the cell above, and a probability of 0.1 of bumping into the wall and remaining in its present cell.

$$expected\ reward = 0.8 * (-1) + 0.1 * (-3) + 0.1 * (-3)$$

$$expected\ reward = -1.4$$

All of the expected rewards are calculated in this way, taking into account the transition model for this particular robot.

You may have noticed that a few expected rewards are missing in the image above. Can you calculate their values?

Expected Reward Quiz

What is the expected reward for moving from the bottom-left cell to the cell above it?

-1.2 RESET

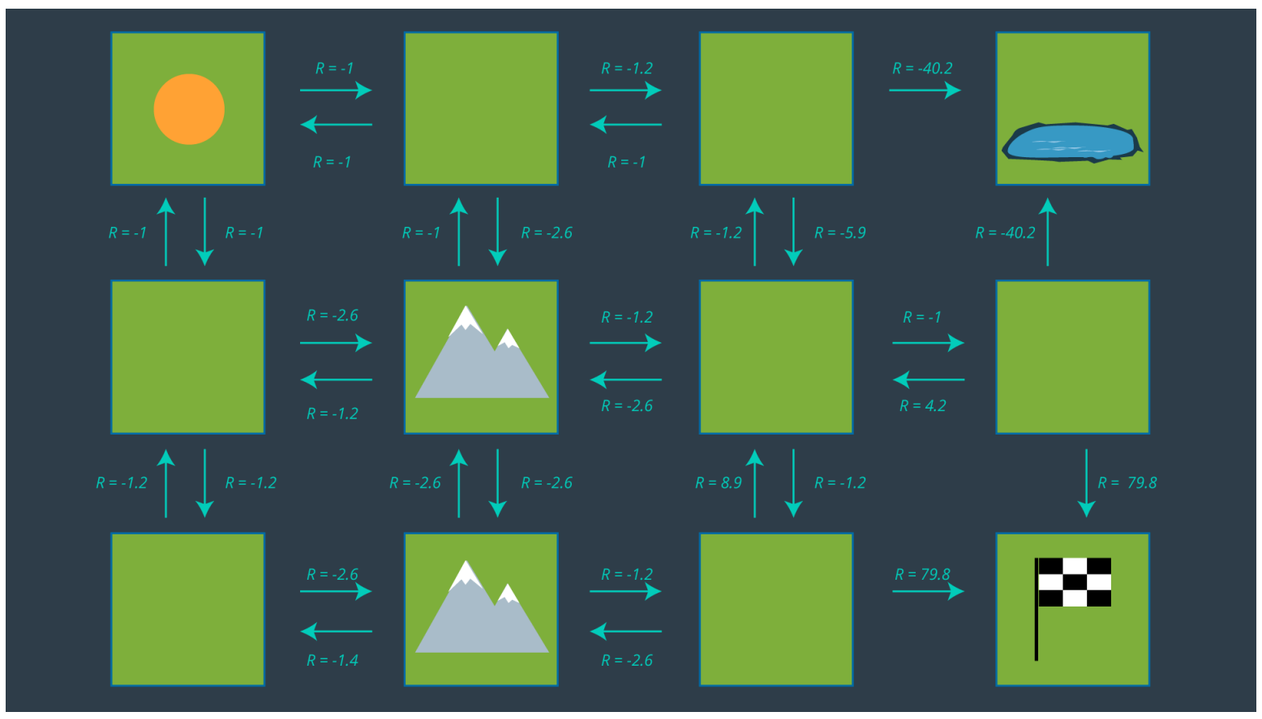
What is the expected reward for moving from the empty cell on the right to the goal,one cell below it?

79.8 RESET

What is the expected reward for moving from the empty cell on the right to the cell to its left?

4.2 RESET

Hopefully, after completing the quizzes, you are more comfortable with how the expected rewards are calculated. The image below has all of the expected rewards filled in.



Selecting a Policy

Now that we have an understanding of our expected rewards, we can select a policy and evaluate how efficient it is. Once again, a policy is just a mapping from states to actions. If we review the set of actions depicted in the image above, and select just one action for each state - i.e. exactly one arrow leaving each cell (with the exception of the hazard and goal states) - then we have ourselves a policy.

However, we're not looking for *any* policy, we'd like to find the *optimal* policy. For this reason, we'll need to study the utility of each state to then determine the *best* action to take from each state. That's what the next concept is all about!

State Utility

Definition

The **utility of a state** (otherwise known as the **state-value**) represents how attractive the state is with respect to the goal. Recall that for each state, the state-value function yields the expected return, if the agent (robot) starts in that state and then follows the policy for all time steps. In mathematical notation, this can be represented as so:

$$U^{\pi}(s) = E[\sum_{t=0}^{\infty} R(s_t)|\pi, s_0 = s]$$

The notation used in path planning differs slightly from what you saw in Reinforcement Learning. But the result is identical.

Here,

- $U^{\pi}(s)$ represents the utility of a state s,
- E represents the *expected* value, and
- $R(s)$ represents the reward for state s.

The utility of a state is the sum of the rewards that an agent would encounter if it started at that state and followed the policy to the goal.

Calculation

We can break the equation down, to further understand it.

$$U^{\pi}(s) = E[\sum_{t=0}^{\infty} R(s_t)|\pi, s_0 = s]$$

Let's start by breaking up the summation and explicitly adding all states.

$$U^{\pi}(s) = E[R(s_0) + R(s_1) + R(s_2) + ... |\pi, s_0 = s]$$

Then, we can pull out the first term. The expected reward for the first state is independent of the policy. While the expected reward of all future states (those between the state and the goal) depend on the policy.

$$U^{\pi}(s) = E[R(s_0)|s_0 = s] + E[R(s_1) + R(s_2) + ... |\pi]$$

Re-arranging the equation results in the following. (Recall that the prime symbol, as on s', represents the next state - like s_2 would be to s_1).

$$U^{\pi}(s) = R(s) + E[\sum_{t=0}^{\infty} R(s_t)|\pi, s_0 = s']$$

Ultimately, the result is the following.

$$U^{\pi}(s) = R(s) + U^{\pi}(s')$$

As you see here, calculating the utility of a state is an iterative process. It involves all of the states that the agent would visit between the present state and the goal, as dictated by the policy.

As well, it should be clear that the utility of a state depends on the policy. If you change the policy, the utility of each state will change, since the sequence of states that would be visited prior to the goal may change.

Determining the Optimal Policy

Recall that the **optimal policy**, denoted π^*, informs the robot of the *best* action to take from any state, to maximize the overall reward. That is,

$$\pi^*(s) = \underset{a}{argmax}\ E[U^{\pi}(s)]$$

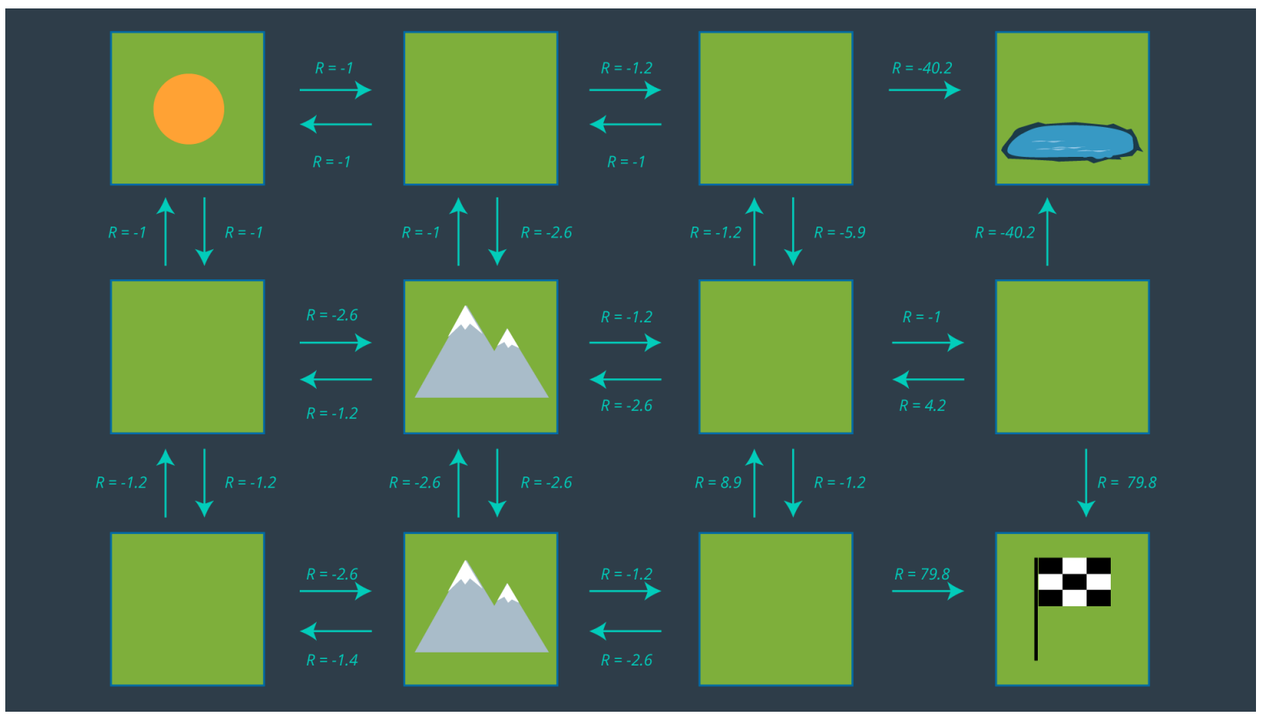
In a state s, the optimal policy π^* will choose the action a that maximizes the utility of s (which, due to its iterative nature, maximizes the utilities of all future states too).

While the math may make it seem intimidating, it's as easy as looking at the set of actions and choosing the best action for every state. The image below displays the set of all actions once more.

$$a$$

In a state s, the optimal policy π^* will choose the action a that maximizes the utility of s (which, due to its iterative nature, maximizes the utilities of all future states too).

While the math may make it seem intimidating, it's as easy as looking at the set of actions and choosing the best action for every state. The image below displays the set of all actions once more.



It may not be clear from the get-go which action is optimal for every state, especially for states far away from the goal which have many paths available to them. It's often helpful to start at the goal and work your way backwards.

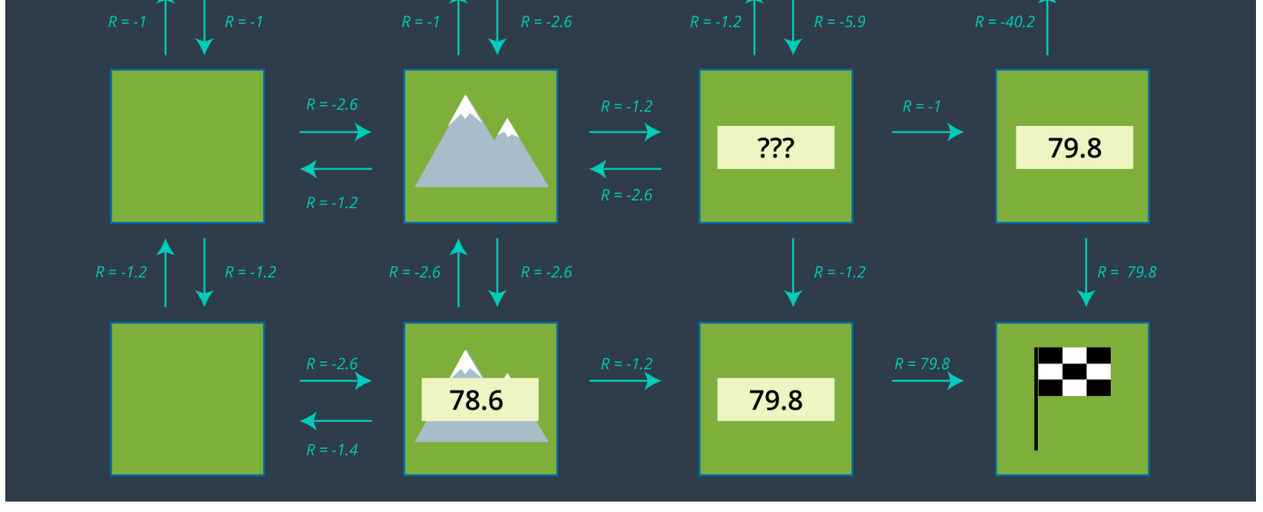
If you look at the two cells adjacent to the goal, their best action is trivial - go to the goal! Recall from your learning in RL that the goal state's utility is 0. This is because if the agent starts at the goal, the task is complete and no reward is received. Thus, the expected reward from either of the goal's adjacent cells is 79.8. Therefore, the state's utility is, 79.8 + 0 = 79.8 (based on $U^{\pi}(s) = R(s) + U^{\pi}(s')$).

If we look at the lower mountain cell, it is also easy to guess which action should be performed in this state. With an expected reward of -1.2, moving right is going to be much more rewarding than taking any indirect route (up or left). This state will have a utility of -1.2 + 79.8 = 78.6.

Now it's your turn!

Quiz

Can you calculate what would the utility of the state to the right of the center mountain be, if the most rewarding action is chosen?



What is the utility of the state to the right of the center mountain following the optimal policy?

Enter your response here

What is the utility of the state to the right of the center mountain following the optimal policy?

Enter your response here

SUBMIT

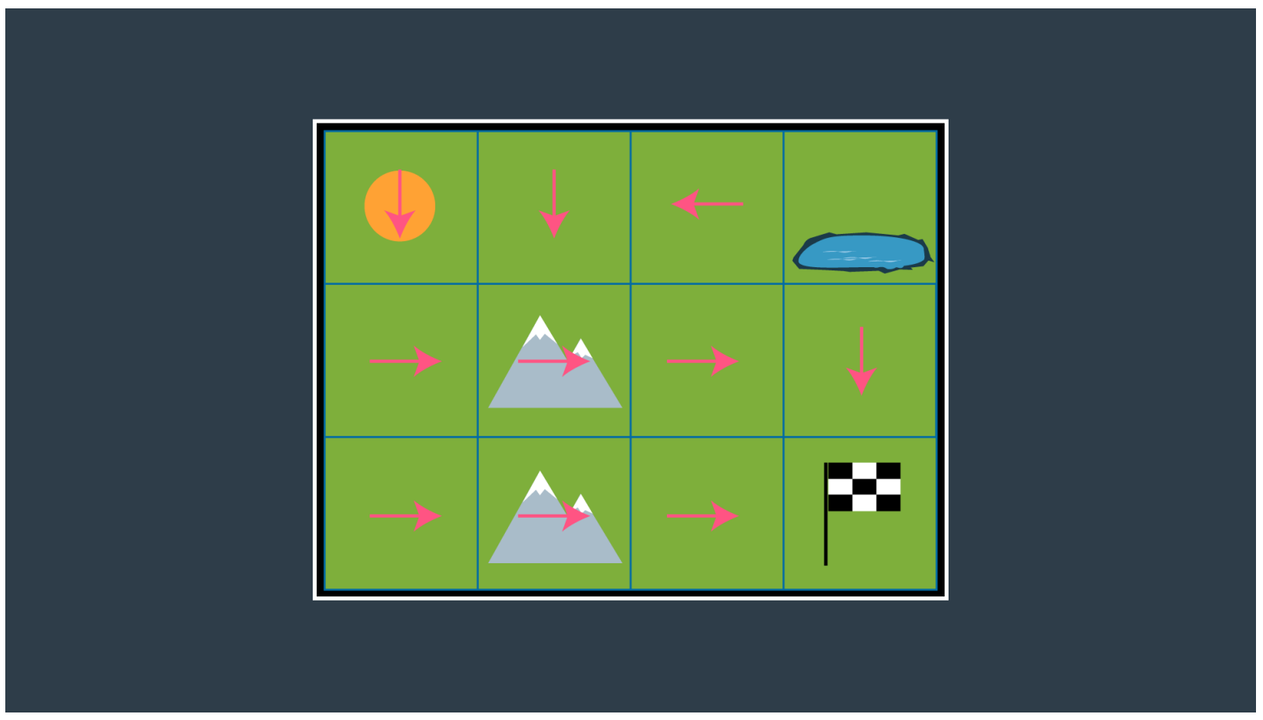
The process of selecting each state's most rewarding action continues, until every state is mapped to an action. These mappings are precisely what make up the policy.

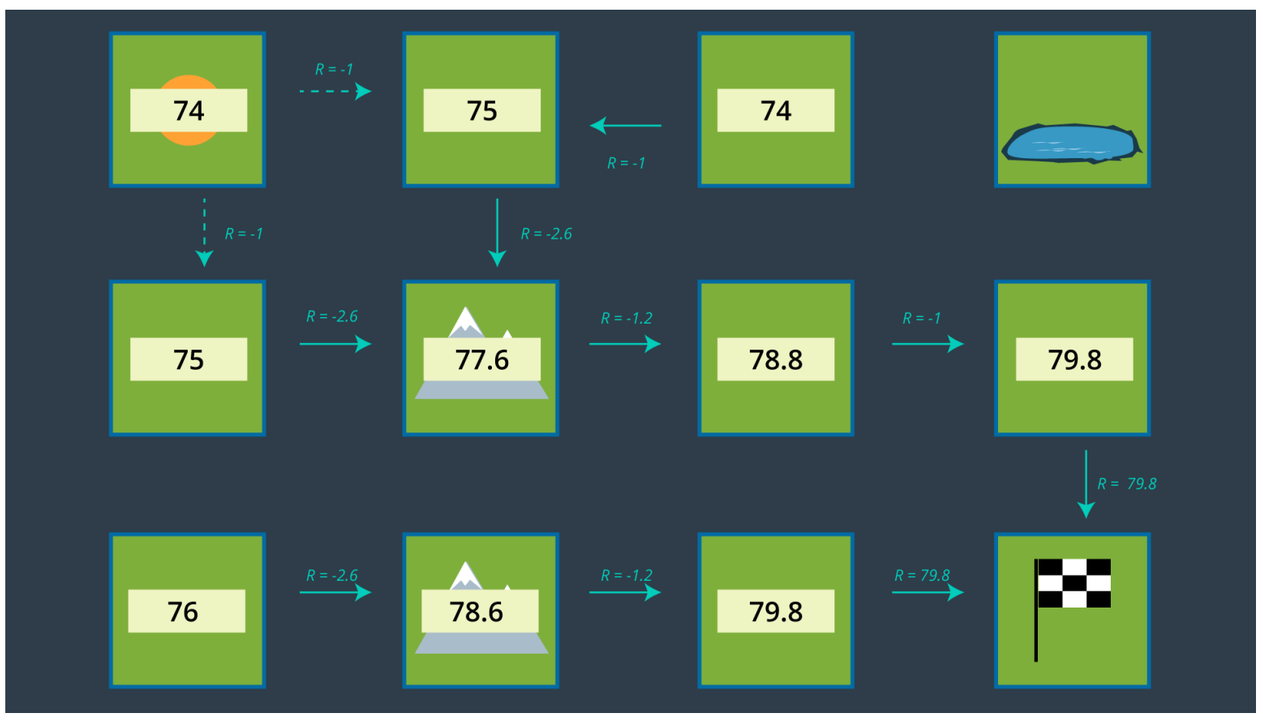
It is highly suggested that you pause this lesson here, and work out the optimal policy on your own using the action set seen above. Working through the example yourself will give you a better understanding of the challenges that are faced in the process, and will help you remember this content more effectively. When you are done, you can compare your results with the images below.

Applying the Policy

Once this process is complete, the agent (our robot) will be able to make the best path planning decision from every state, and successfully navigate the environment from any start position to the goal. The optimal policy for this environment and this robot is provided below.

The image below that shows the set of actions with just the optimal actions remaining. Note that from the top left cell, the agent could either go down or right, as both options have equal rewards.





Discounting

One simplification that you may have noticed us make, is omit the discounting rate γ. In the above example, $\gamma = 1$ and all future actions were considered to be just as significant as the present action. This was done solely to simplify the example. After all, you have already been introduced to γ through the lessons on Reinforcement Learning.

In reality, discounting is often applied in robotic path planning, since the future can be quite uncertain. The complete equation for the utility of a state is provided below:

$$U^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi, s_0 = s]$$

NEXT

Value Iteration Algorithm

The process that we went through to determine the optimal policy for the mountainous environment was fairly straightforward, but it did take some intuition to identify which action was optimal for every state. In larger more complex environments, intuition may not be sufficient. In such environments, an algorithm should be applied to handle all computations and find the optimal solution to an MDP. One such algorithm is called the Value Iteration algorithm. *Iteration* is a key word here, and you'll see just why!

The Value Iteration algorithm will initialize all state utilities to some arbitrary value - say, zero. Then, it will iteratively calculate a more accurate state utility for each state, using
$U(s) = R(s) + \gamma max_a \Sigma_{s'} T(s, a, s') U(s')$

Algorithm

> $U' = 0$
>
> loop until $close\text{-}enough(U, U')$
>
> > $U = U'$
> >
> > for s in S, do:
> >
> > > $U(s) = R(s) + \gamma max_a \Sigma_{s'} T(s, a, s') U(s')$
>
> return U

With every iteration, the algorithm will have a more and more accurate estimate of each state's utility. The number of iterations of the algorithm is dictated by a function $close\text{-}enough$ which detects convergence. One way to accomplish this is to evaluate the root mean square error,

$$RMS = \frac{1}{|S|} \sqrt{\sum_s (U(s) - U'(s))^2}$$

Once this error is below a predetermined threshold, the result has converged sufficiently.

$$RMS(U, U') < \epsilon$$

This algorithm finds the optimal policy to the MDP, regardless of what U' is initialized to (although the efficiency of the algorithm will be affected by a poor U').