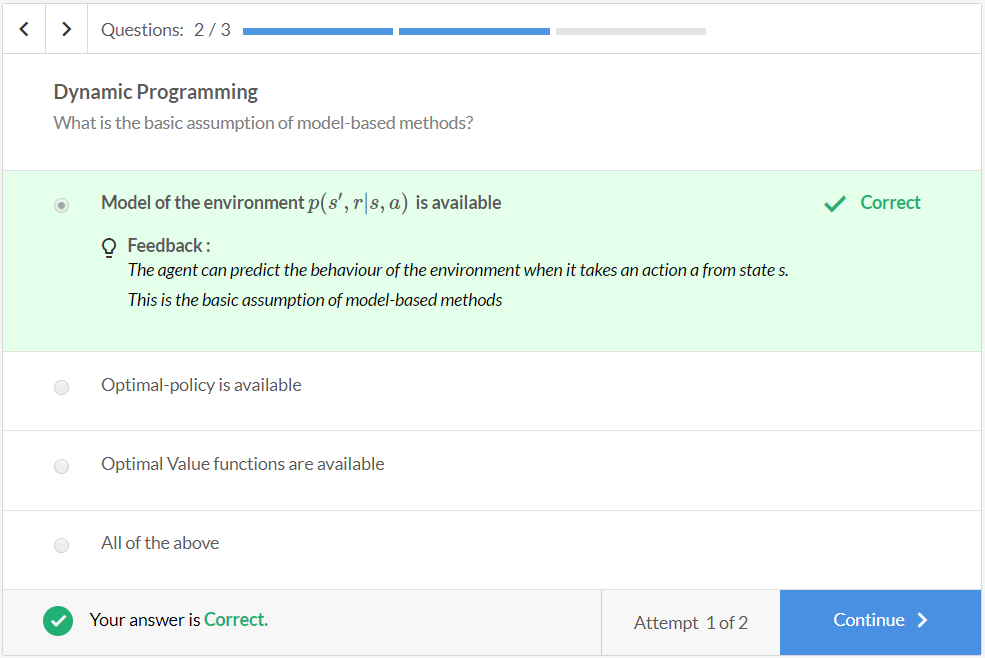
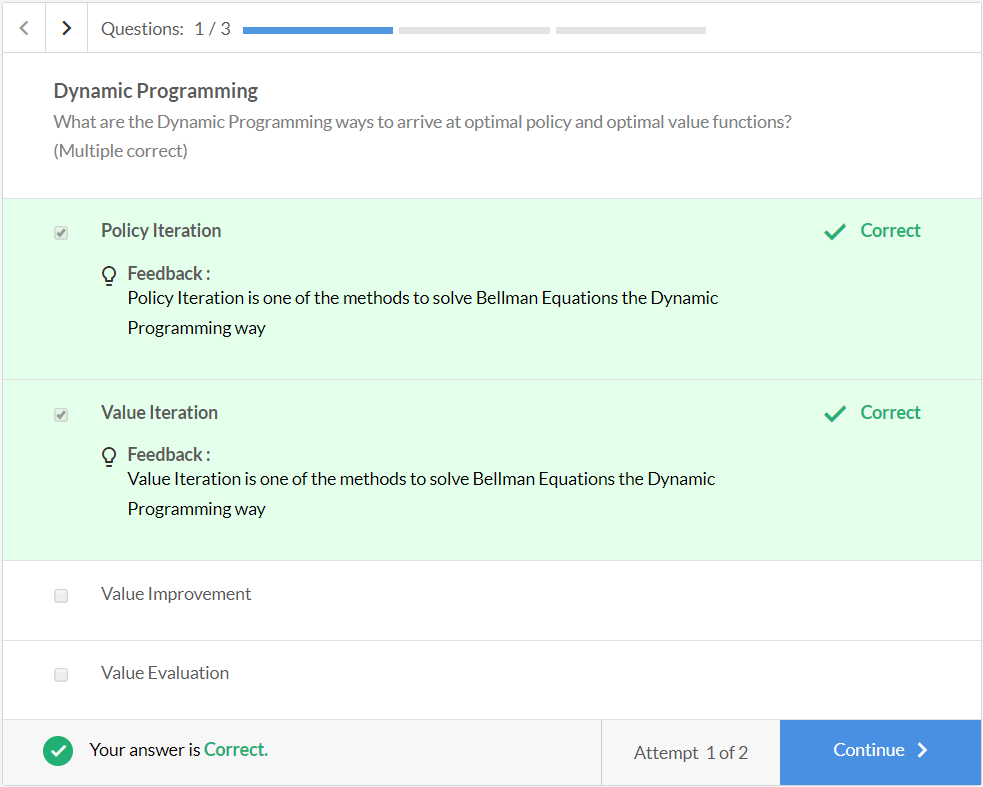
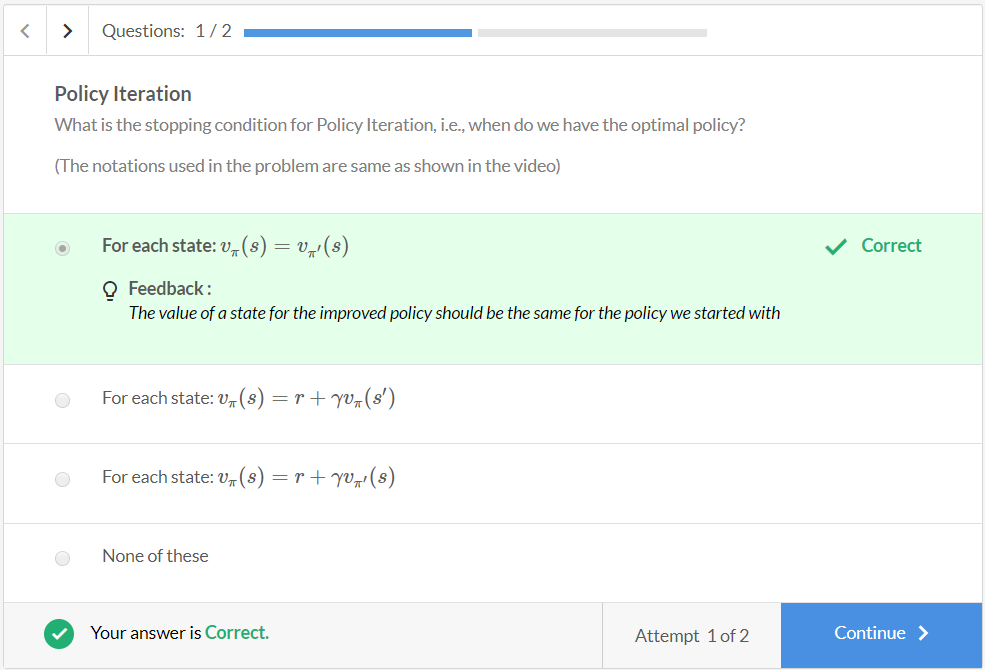
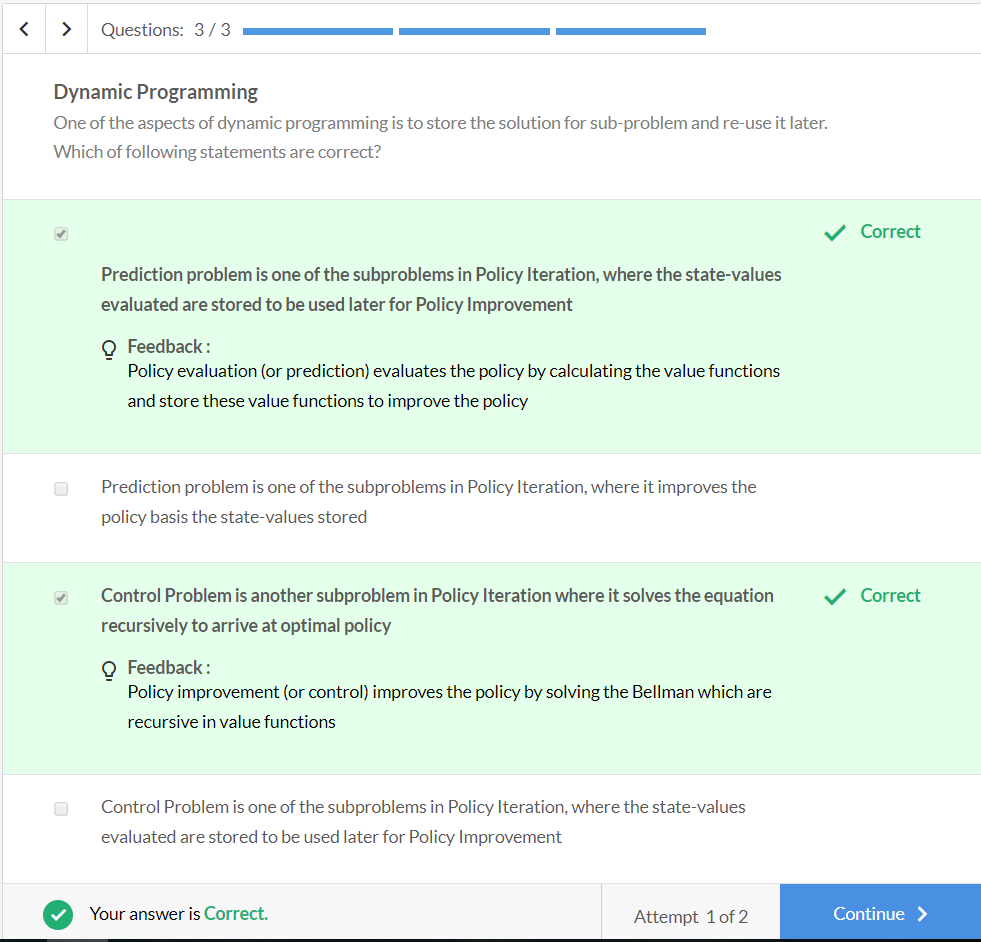
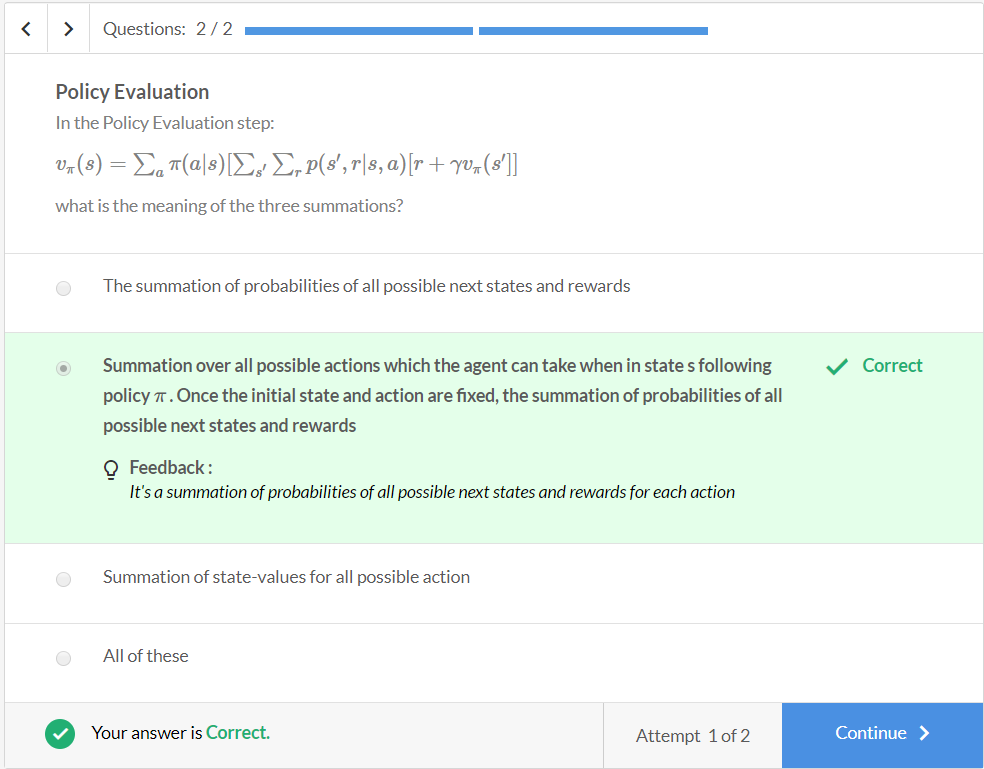
**POLICY Iteration**







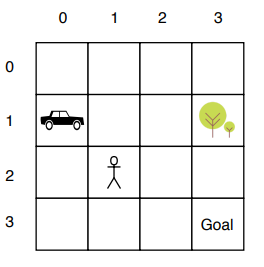
**Policy Iteration - Comprehension**

Consider a self-driving car being trained in a small **4x4** grid. All the obstacles, i.e., the pedestrian and the tree, are stationary. When the car runs into the wall (boundaries of the GridWorld), it ends up at the same location. The objective of the agent is to find an**optimal path** **to the goal** using policy iteration. An episode terminates if the agent runs into an obstacle or reaches the goal. The values of terminal states are 0, i.e., v(s) = 0.

**Assumption**: The environment is deterministic, i.e., given the state and the action, the agent will transition only to one particular state.

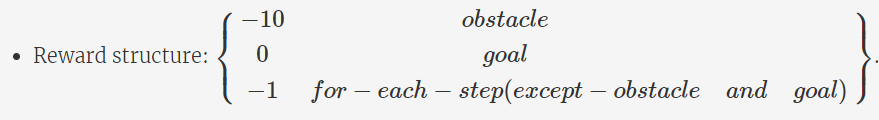
**Formulating the MDP**

The most important step while solving an RL problem is formulating the MDP. You can think of MDP as the simulation of the environment, which decides the consequence of the agent's action. Let's start with it.

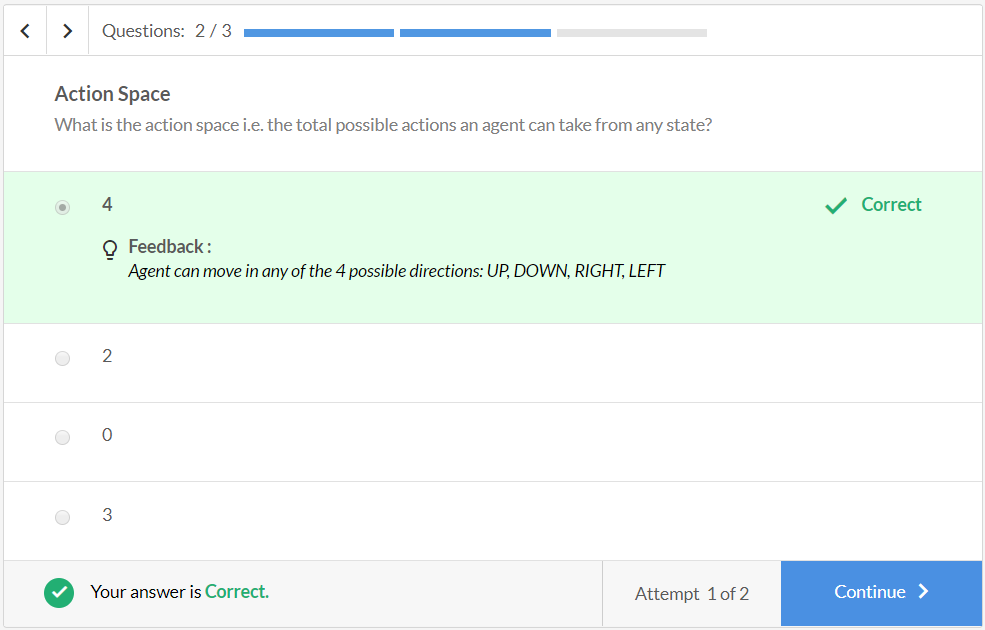
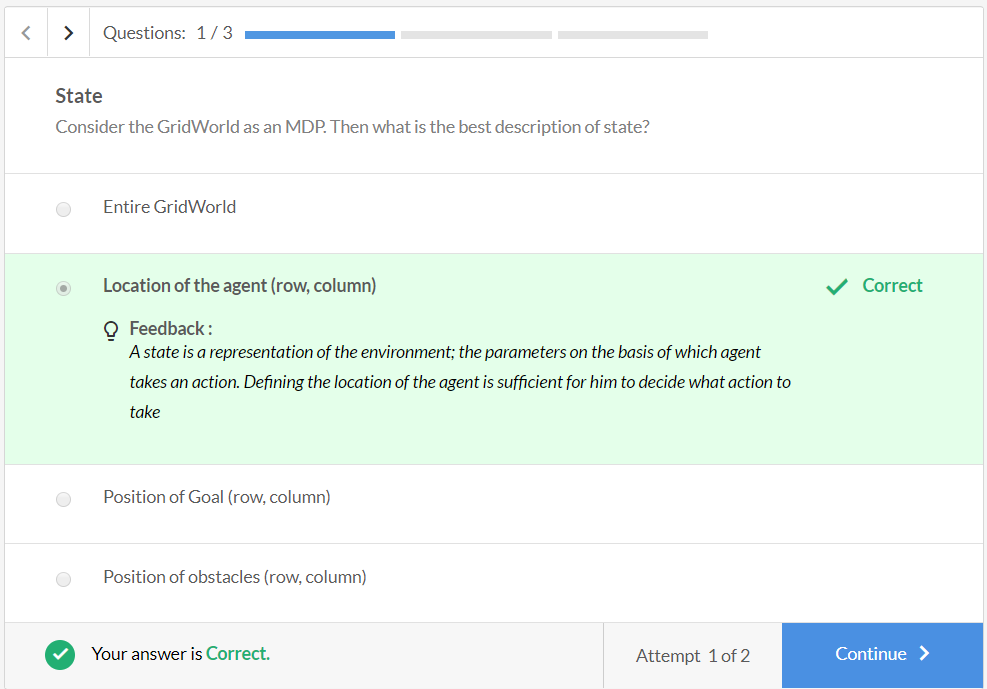


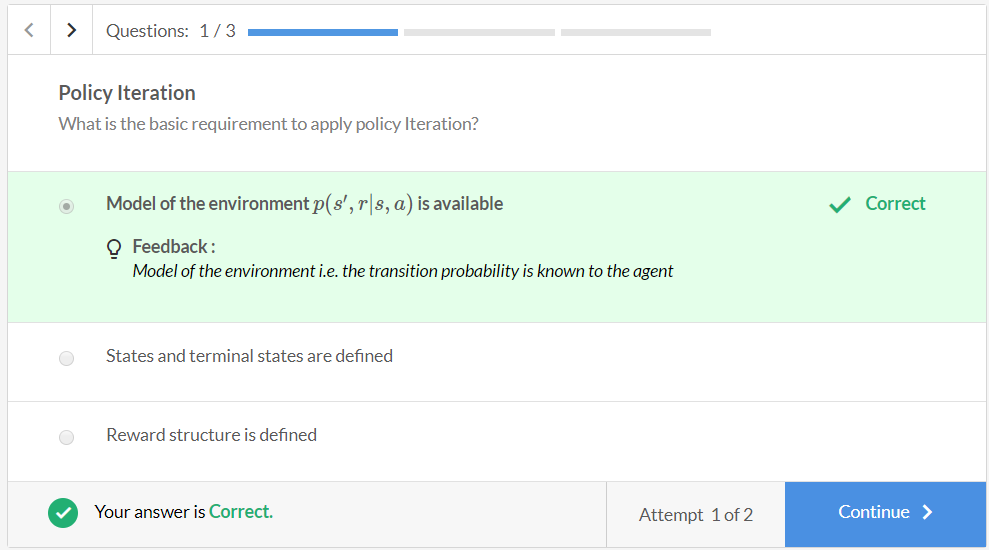
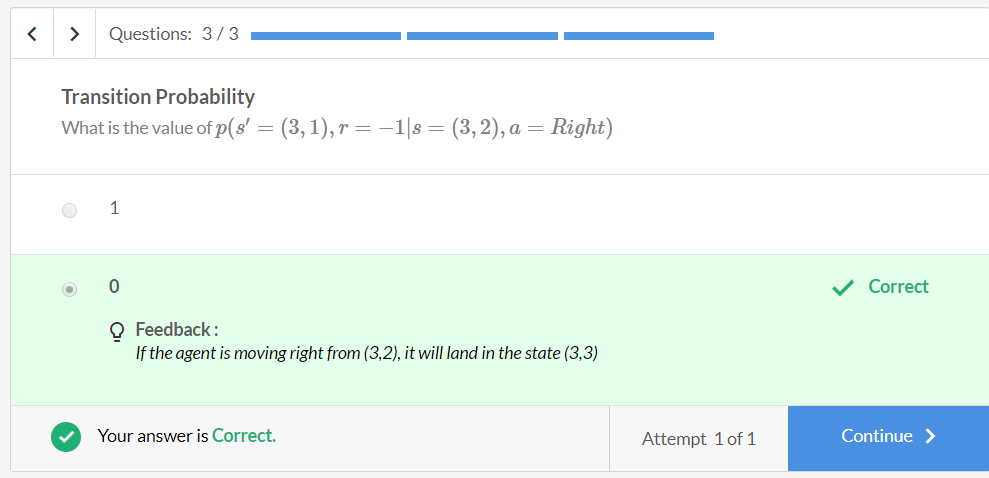
For creating the MDP environment, you need to specify states, actions, reward structure, transition probabilities and terminal states (to decide when the episode will end).

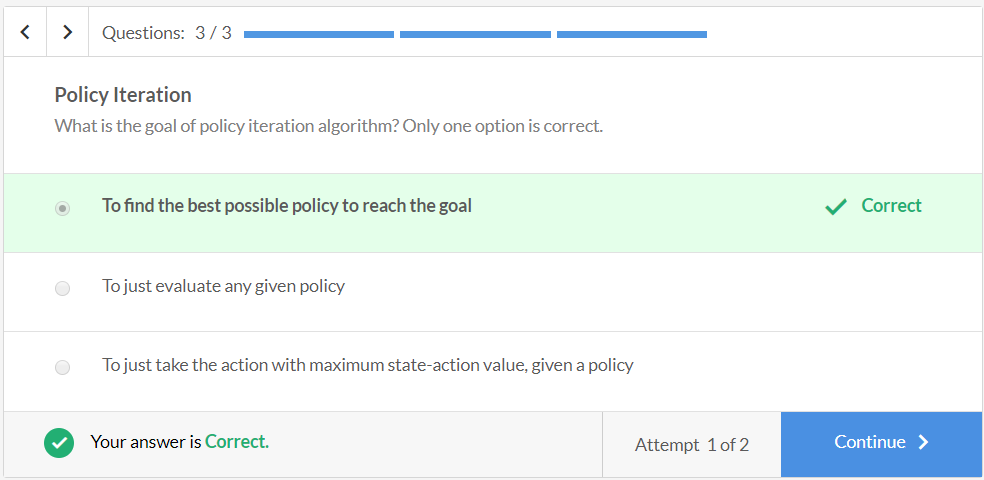
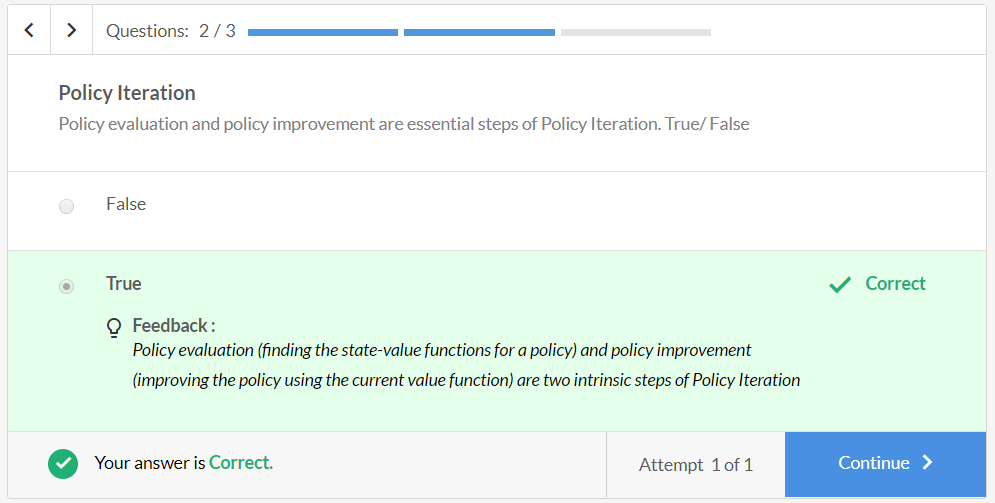
Please note that the state here is represented as (row no., column no.)

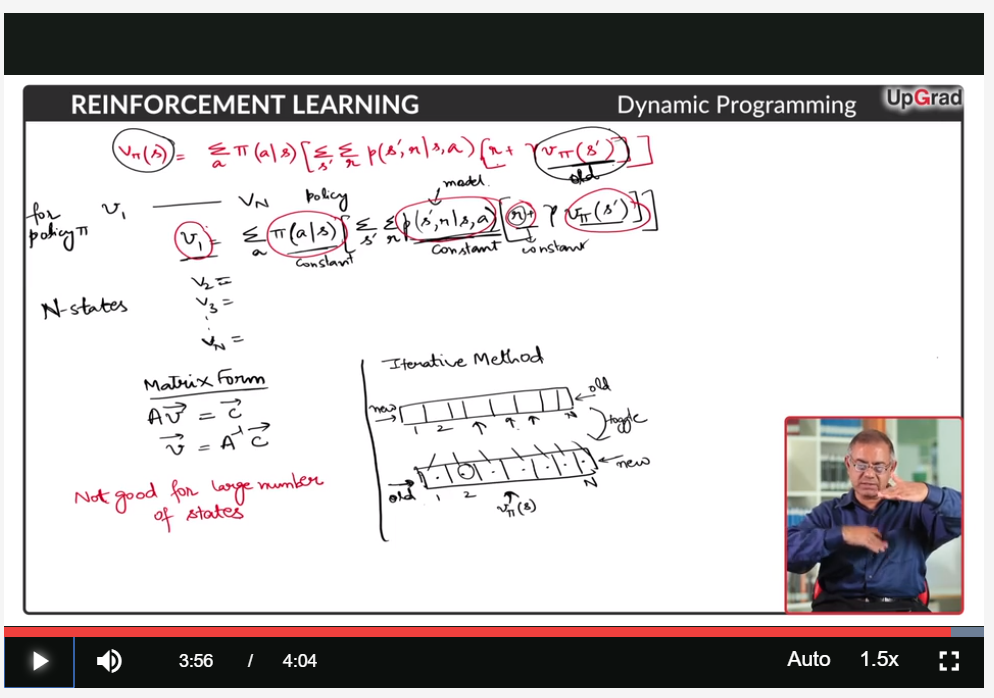
* Action space for the agent: {UP, DOWN, RIGHT, LEFT}
* The reward structure is designed like this because there is a huge negative reward for an obstacle, and a -1 for each step makes sure that the agent learns to reach the goal in the minimum number of steps. The reward for the goal is 0, which is fine because it is more than reaching any other state.
* States: the position of agent in the grid
* Terminal states: Obstacles (pedestrian, tree) & Goal. The state-values of terminal states is 0.
* Transition probability for each state-action pair (let's define a few to get the intuition): Let's say the car is at (1,0) and it takes an action 'UP', it will move to (0,0) and get a reward of -1. Also, it is certain that it will get a reward of -1, so the probability is 1. Hence, p(s′=(0,0),r=−1|s=(1,0),a=Up)=1
  + Similarly, let's define a few more probabilities:
    - p(s′=(3,2),r=−1|s=(2,2),a=Down)=1
    - p(s′=(1,3),r=−10|s=(1,2),a=Right)=1
    - p(s′=(0,2),r=−1|s=(0,2),a=Up)=1
    - p(s′=(1,1),r=−10|s=(1,2),a=Right)=0

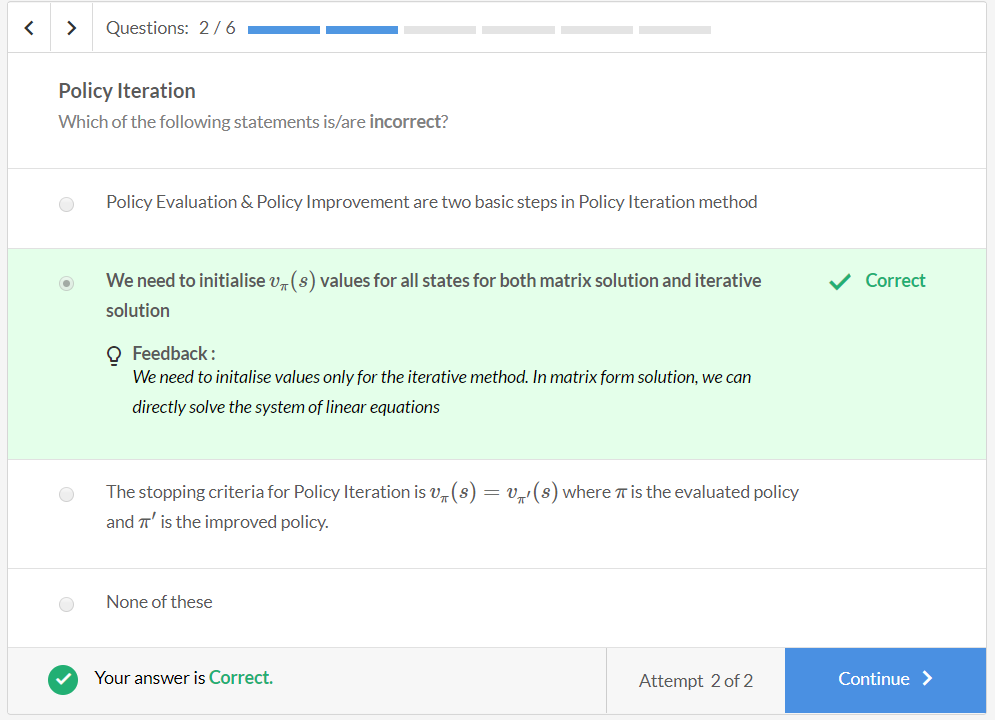
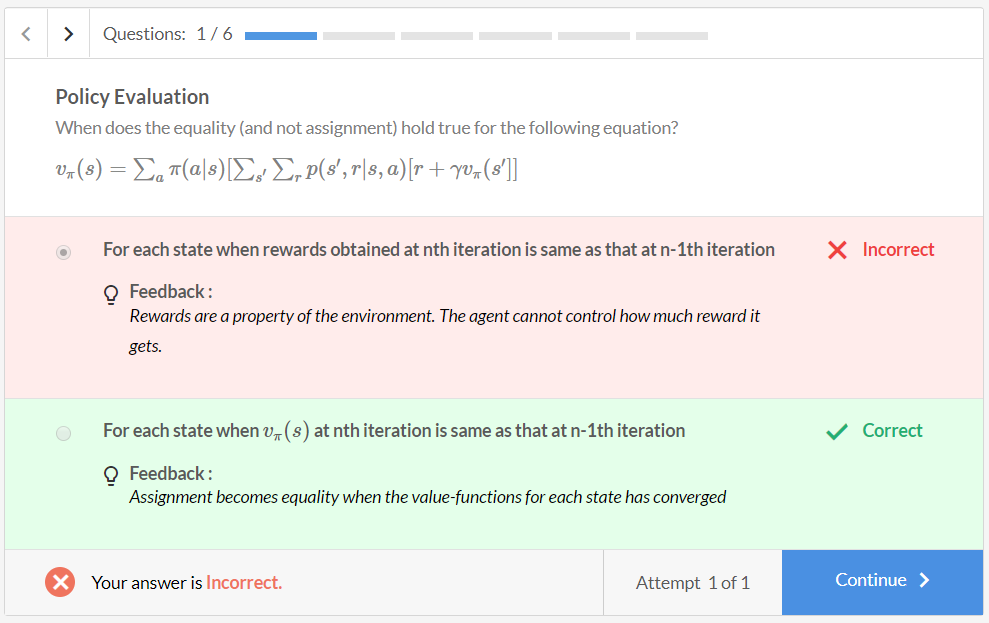
Note that these transition probabilities basically represent the model of the environment.

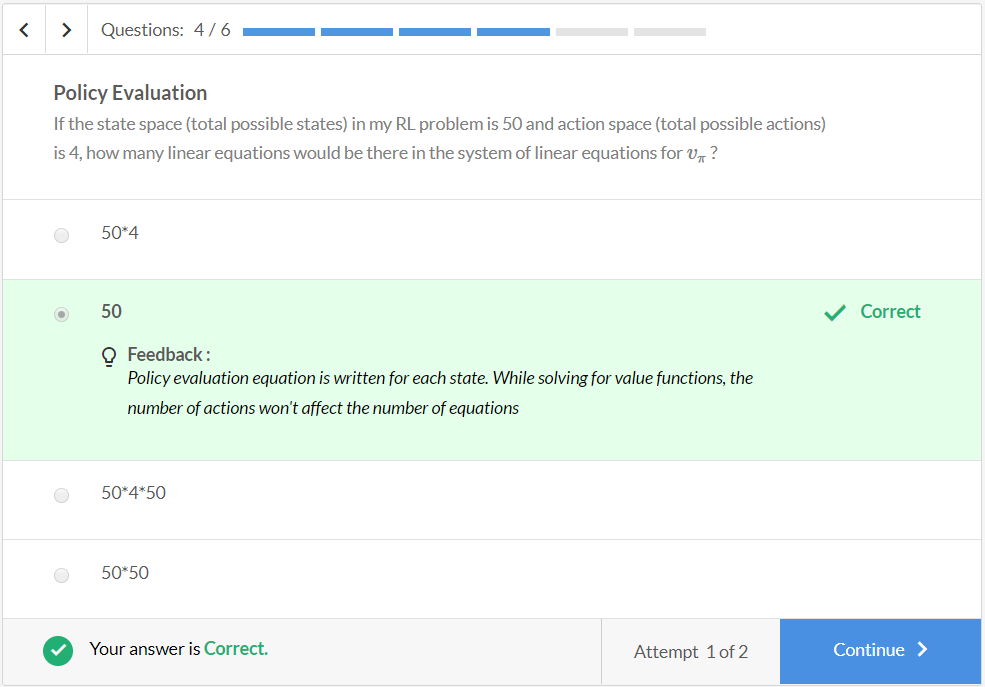
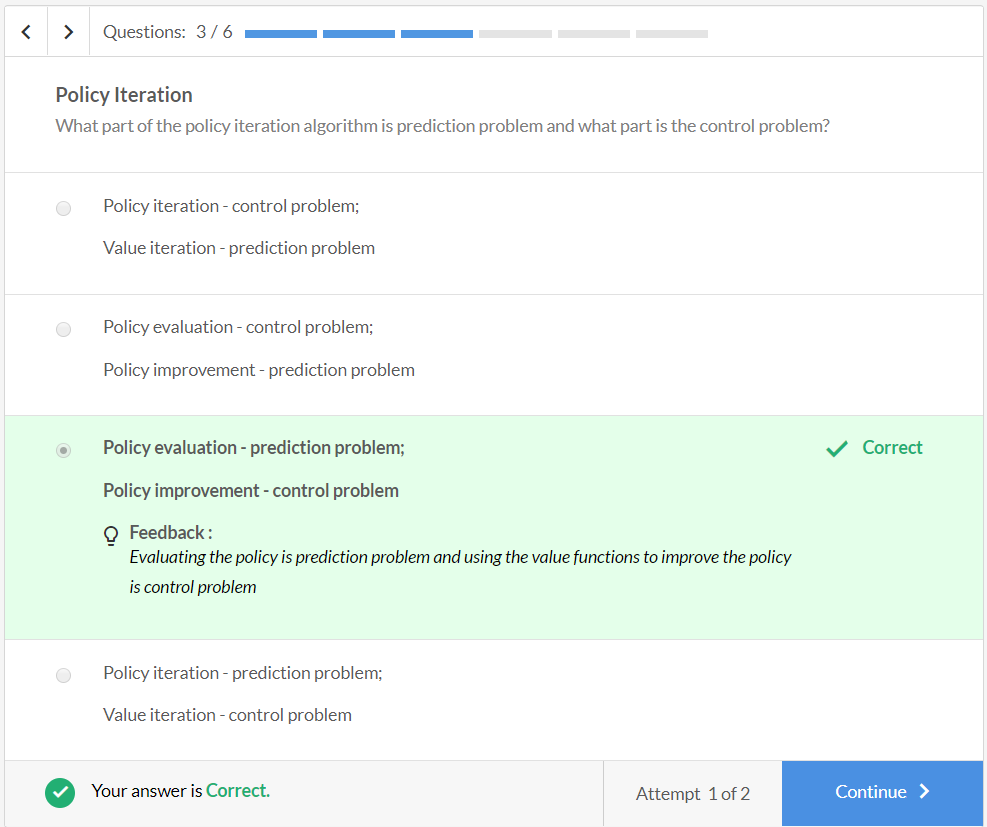


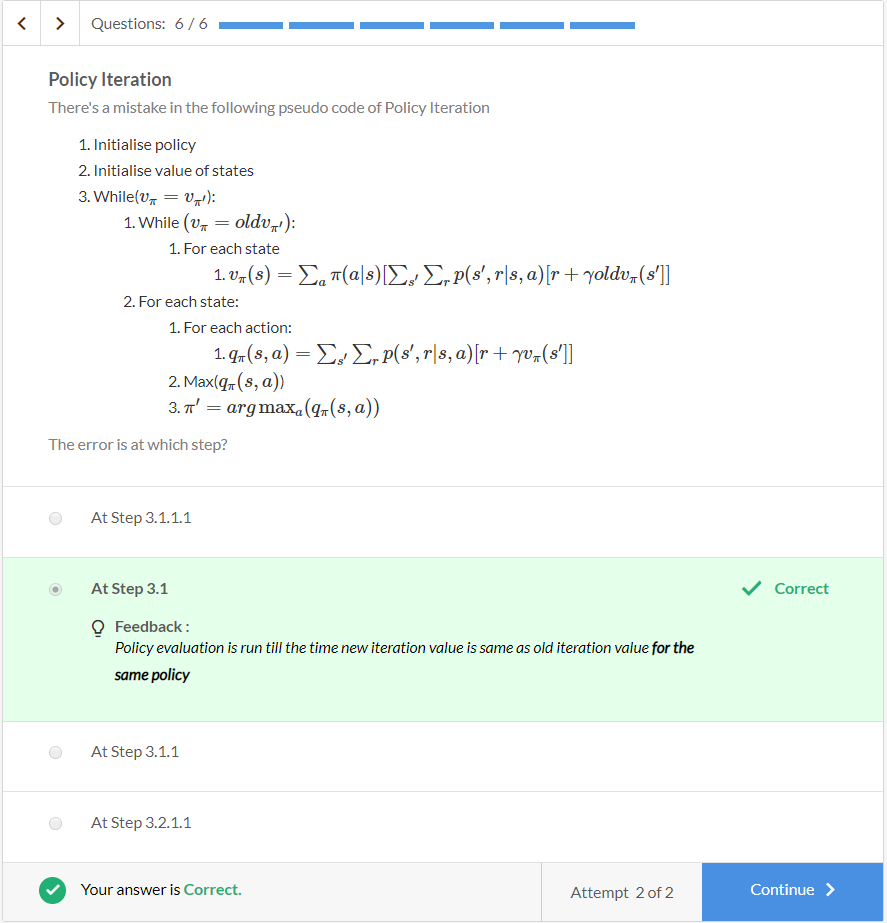
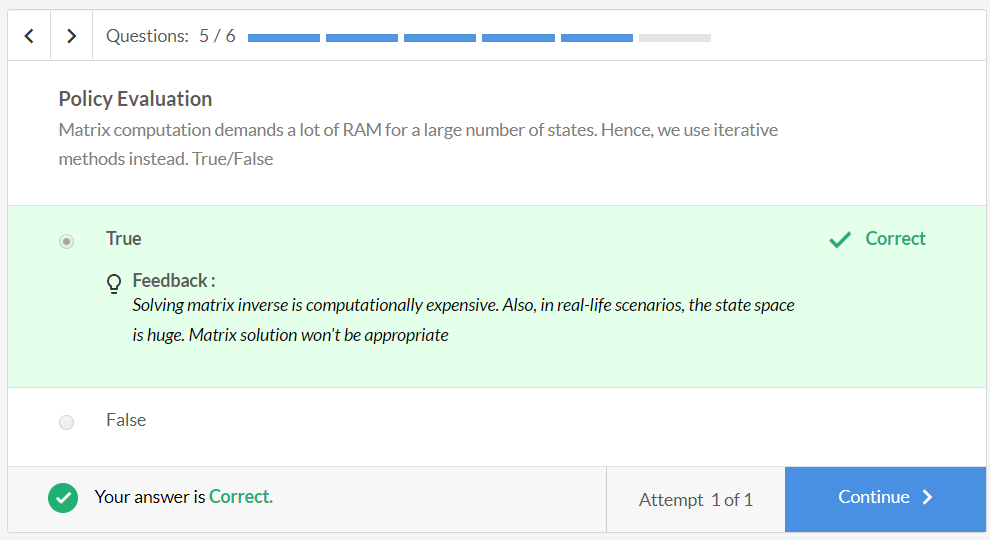












## Example - GridWorld

Let's try to understand the policy improvement procedure through the following example.

Consider this small **3x3** grid. Say for some arbitrary policy π; the state-value functions are as follows (the numbers in the grid represent the state-values of the corresponding state):

| GridWorld | | | |
| --- | --- | --- | --- |
| **Grid** | **0** | **1** | **2** |
| **0** | 5 | 10 | 20 (Destination) |
| **1** | -10 | 7 | 0 |
| **2** | 9 | -5 | -10 |

The leftmost column and the topmost row are the grid positions. For example, vπ(s=(0,0))=5. Assume that there is a reward of -1 if you take a step, in any direction.

Now, there are two possible actions that can be performed at (0,0) - go **right** or go **down** (rest are the walls). Let's say that the current policy says that one should go down when in the state (0,0). Let's apply policy improvement and see whether the policy actually improves (assume γ=1).

The state-action value for **RIGHT** action is:

qπ(s=(0,0),a=Right)=∑s′∑r[r+γvπ(s′=(0,1))]

=1∗[−1+1∗10]=**9**

The state-action value for **DOWN** action is:

qπ(s=(0,0),a=Down)=∑s′∑r[r+γvπ(s′=(1,0))]

=1∗[−1+1∗(−10)]=**−11**

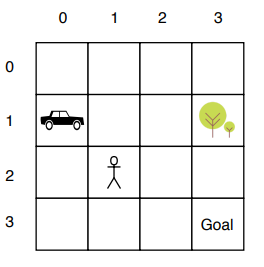
So, the best action, in this case, is to move RIGHT. The improved policy for s=(0,0) will be to move RIGHT, instead of going DOWN as according to the current policy. Similarly, you do the policy improvement for all the states.

On the next page, you will perform both the steps of policy iteration on a GridWorld problem. This will serve as a very good example to help you grasp the concept thoroughly.

**Policy Iteration - GridWorld**

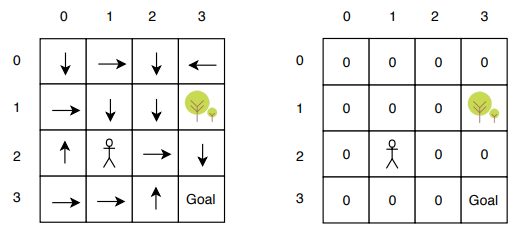
Let's learn to find the optimal policy using the policy iteration algorithm using the **GridWorld**setting. The agent is again the self-driving car. Please note that the state is represented as (row no., column no.).

The GridWorld is the same.



* Action space for the agent: {UP, DOWN, RIGHT, LEFT}
* Reward structure: ⎧⎪⎨⎪⎩−10obstacle0goal−1for−each−step(except−obstacleandgoal)⎫⎪⎬⎪⎭
* States: the position of agent in the grid
* Terminal states: Obstacles (pedestrian, tree) & Goal. The state-values of terminal states is 0.
* Transition probability for each state-action pair (let's define for a few to get the intuition): Say, the car is at (1,0) and it takes an action 'UP', it will move to (0,0) and get a reward of -1, all this with the probability 1. Hence, p(s′=(0,0),r=−1|s=(1,0),a=Up)=1
  + Similarly, let's define this probability for a few:
    - p(s′=(3,2),r=−1|s=(2,2),a=Down)=1
    - p(s′=(1,3),r=−10|s=(1,2),a=Right)=1
    - p(s′=(0,2),r=−1|s=(0,2),a=Up)=1
    - p(s′=(1,1),r=−10|s=(1,2),a=Right)=0

Say you **initialise the policy** π and **state-value** functions v(s) as shown in the figure below. The locations of the obstacles and the goal are fixed. The obstacles here are the tree and the pedestrian. The state-values for obstacles and the goal are fixed.



The **arrows** represent the policy. For example, at state (1,2), the policy is to go Down. Also, let's initialise the state-value function for each state to 0, as shown in the right-hand side figure.

keyboard\_arrow\_leftkeyboard\_arrow\_rightQuestions:1/3

**Policy Iteration**

The initialised policy is stochastic. True/False

Top of Form



True



False

Bottom of Form

Submit

Attempt 1 of 1

**Policy Evaluation**

Let's start with policy evaluation. You need to compute vπ(s) for 13 states. Let's start with the first iteration of policy evaluation. Recall that:

                       vπ(s)=∑aπ(a|s)[∑s′∑rp(s′,r|s,a)[r+γvπ(s′)]]

keyboard\_arrow\_leftkeyboard\_arrow\_rightQuestions:1/5

**Policy Evaluation**

Calculate the new value of vπ(s=(3,1)) using the Policy Evaluation equation. Assume γ=1

Top of Form



-4



-1



-1/2



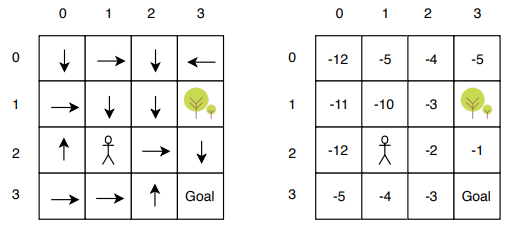
-11

Bottom of Form

Submit

Attempt 1 of 2

Let's say you do this for **n** iterations, where **n** is a fairly large number, and you get a converged value-function for all states for the starting policy as shown below:



Now, to make sure that these are converged values, let's check for one state:

 vπ(s=(0,1))=∑aπ(Right|s=(0,1))[∑s′∑rp(s′,r|s,a)[r+γvπ(s′=(0,2))]]

vπ(s=(0,1))=1∗1∗(−1+1∗(−4))=−5

You can similarly verify for the remaining states.

**Policy Improvement**

Let's now do policy improvement using the current value-estimates.

keyboard\_arrow\_leftkeyboard\_arrow\_rightQuestions:1/2

**Policy Improvement**

Using the converged value-estimates as shown in the figure above, find the improved policy for state =(1,1).

(Hint: You need to evaluate q-value for each action at that state)

Top of Form



Up



Left



Right



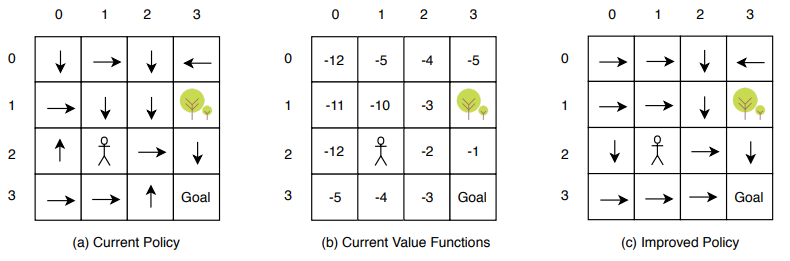
Down

Bottom of Form

Submit

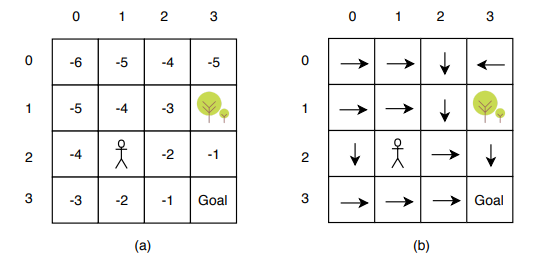
Attempt 1 of 2

After applying policy improvement on all states, you get the following **Improved Policy** (Image (c))



So, this process of **repeated policy evaluation and then policy improvement** is just one step of policy iteration. These steps are run in a loop until you obtain an optimal policy.

Let's say you run these steps n times and you got the below policy and state-values.



keyboard\_arrow\_leftkeyboard\_arrow\_rightQuestions:1/2

**Optimal Policy**

Evaluate value-function for the state (0,0) and (2,2)? Recall that, vπ(s)=∑aπ(a|s)[∑s′∑rp(s′,r|s,a)[r+γvπ(s′)]]

Top of Form



vπ(0,0)=−6;vπ(2,2)=−2



vπ(0,0)=−2;vπ(2,2)=−6

Bottom of Form

Submit

Attempt 1 of 1

Let's understand the results of policy iteration. What did the agent learn?

The optimal policy, in this case, is the ideal direction to move when at some grid position. Say, after training, you put the car at (0,2). It should now know the optimal path to reach the goal since it has learnt the value of each state.

The optimal path would be to move in a **direction where the state-value is better than the current state-value**: (0,2) ->(1,2) ->(2,2) ->(3,2) ->(3,3). So, the car should know what ("optimal") action it should take at every state.

keyboard\_arrow\_leftkeyboard\_arrow\_rightQuestions:1/2

**Policy Iteration**

What is the optimal action from s = (0,3)?

Top of Form



Down



Left



Right



Up

Bottom of Form

Submit

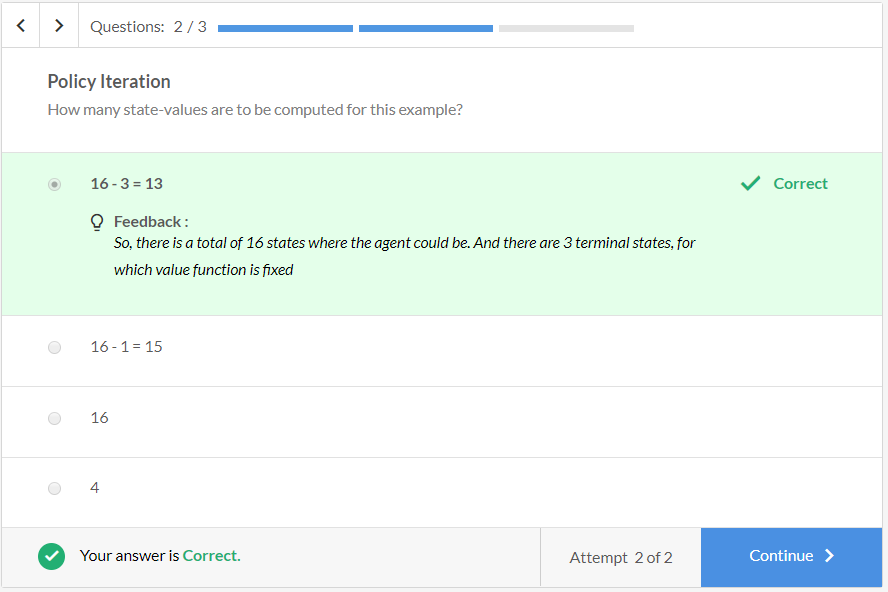
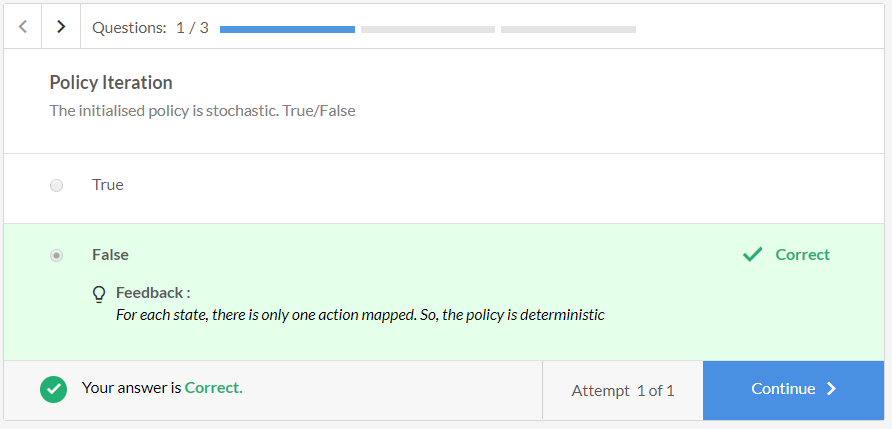
Attempt 1 of 2

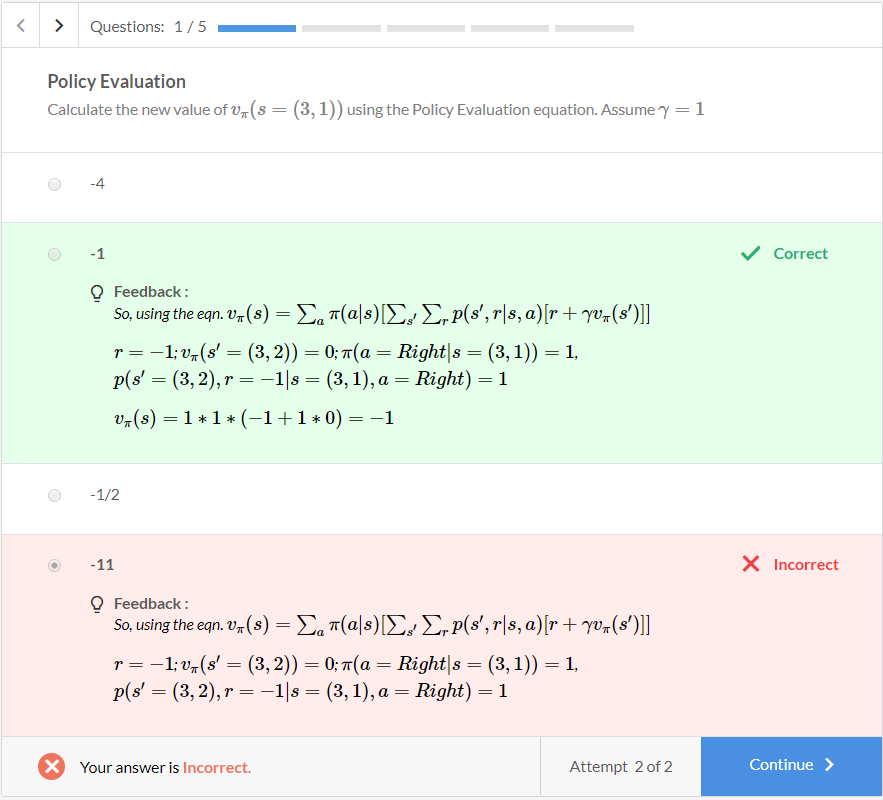
This example illustrates how the policy iteration procedure works. You saw how you can improve a deterministic policy to arrive at an optimal solution.

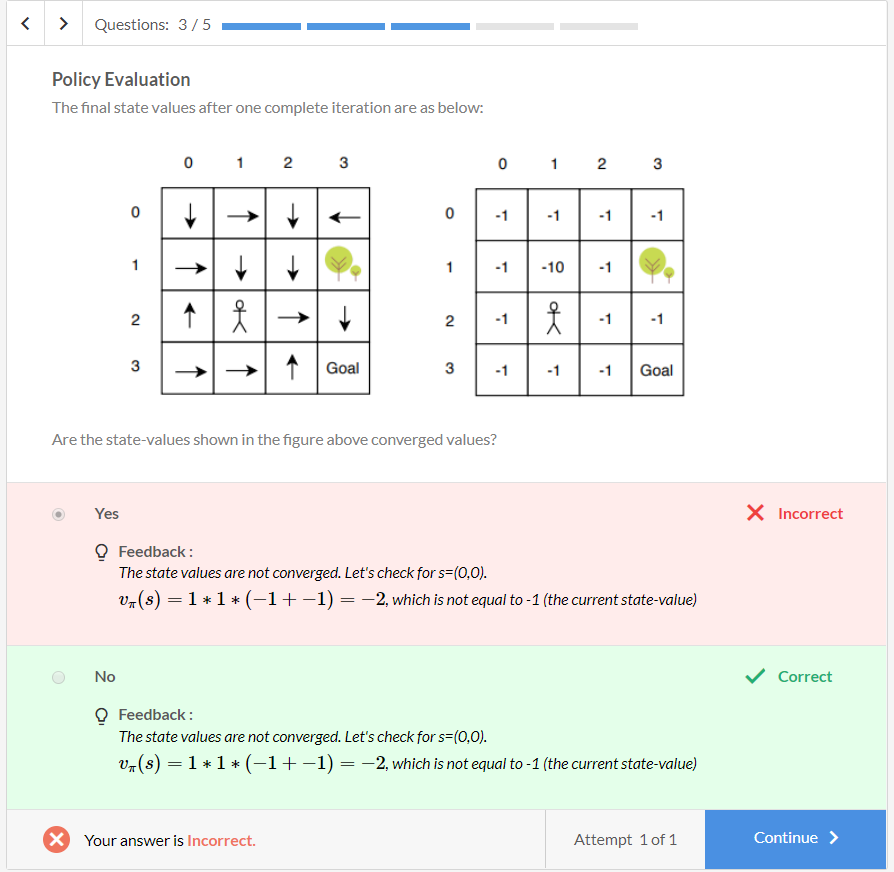
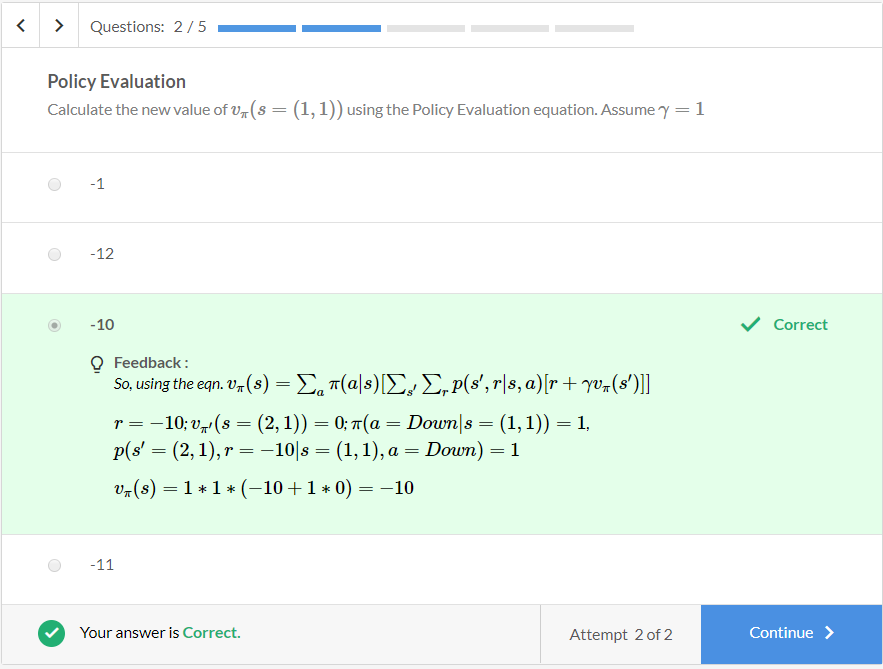
In the next segment, you'll study another algorithm used to learn the optimal policy -**value iteration**.

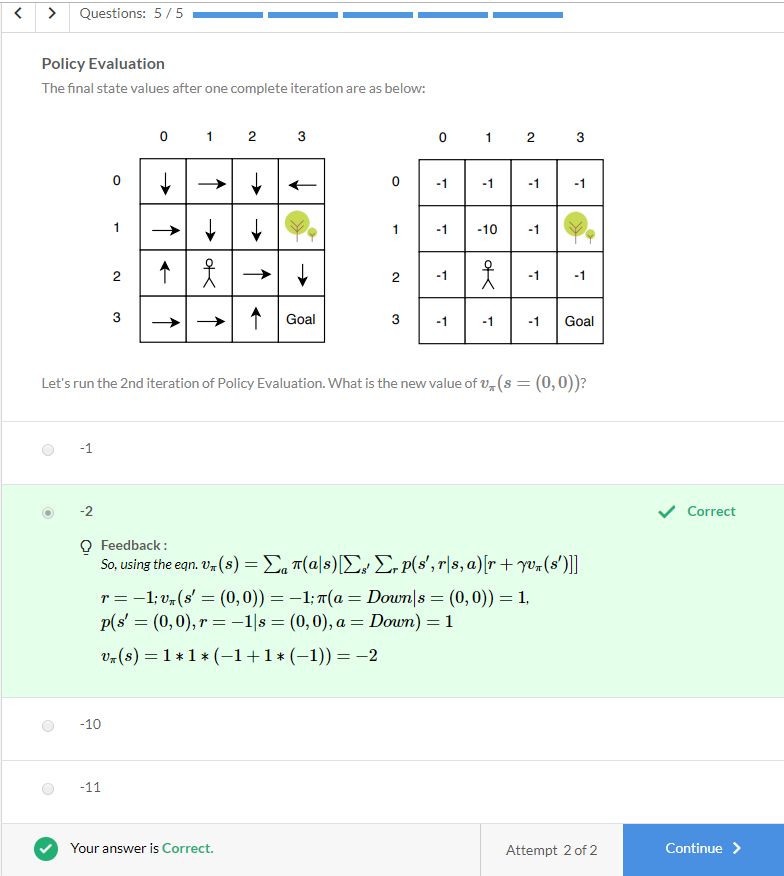
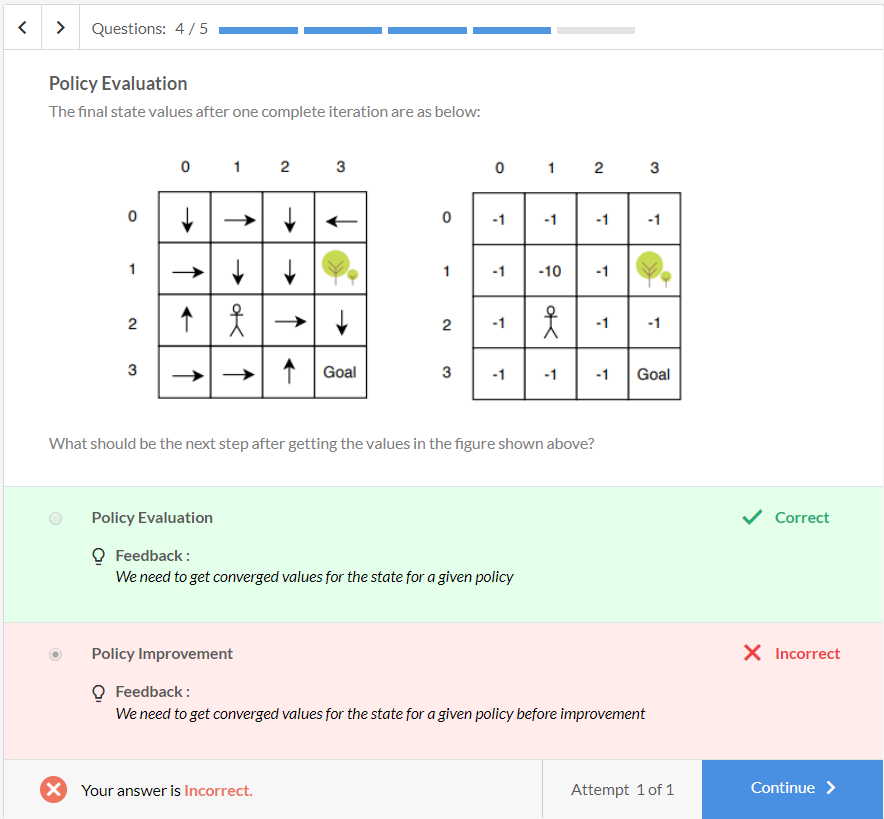
**Policy Iteration for Stochastic Policy - Optional**

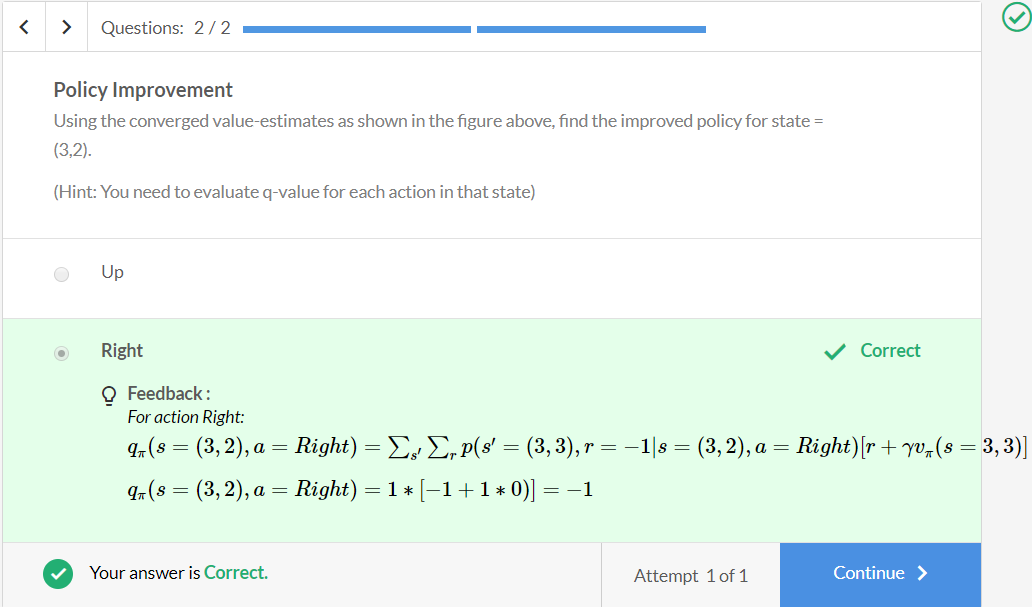
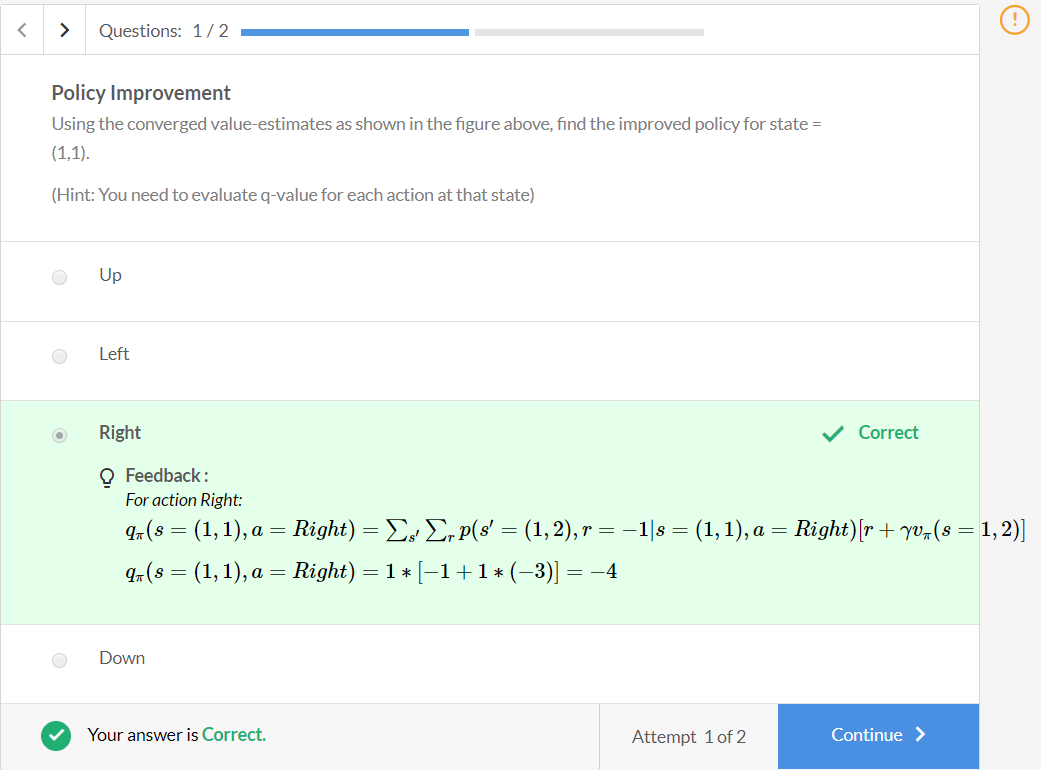
You can start with a stochastic policy as well and follow similar steps to arrive at the optimal policy. We will not cover the stochastic policies in this course. However, you can refer here to see how to solve a similar [GridWorld environment using a stochastic policy](https://learn.upgrad.com/v/course/132/session/26091/segment/134720" \t "_blank). This part is completely optional.

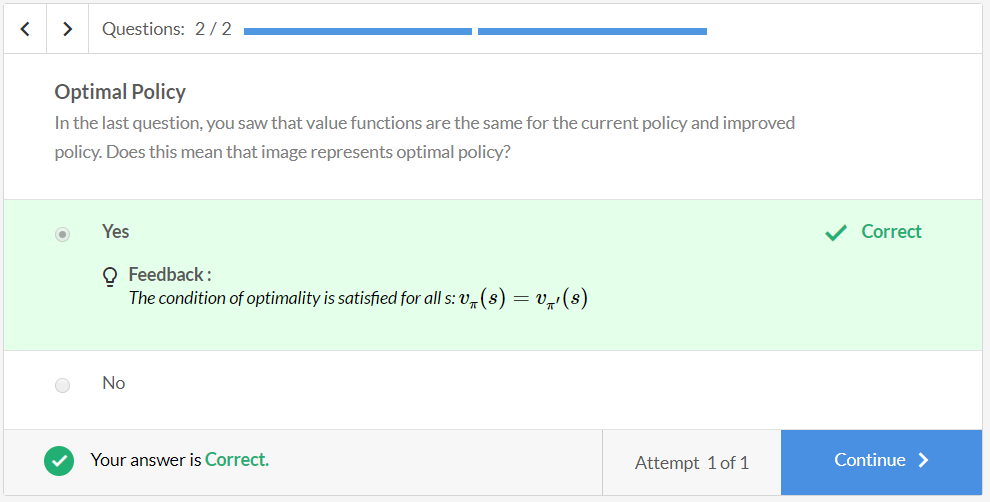
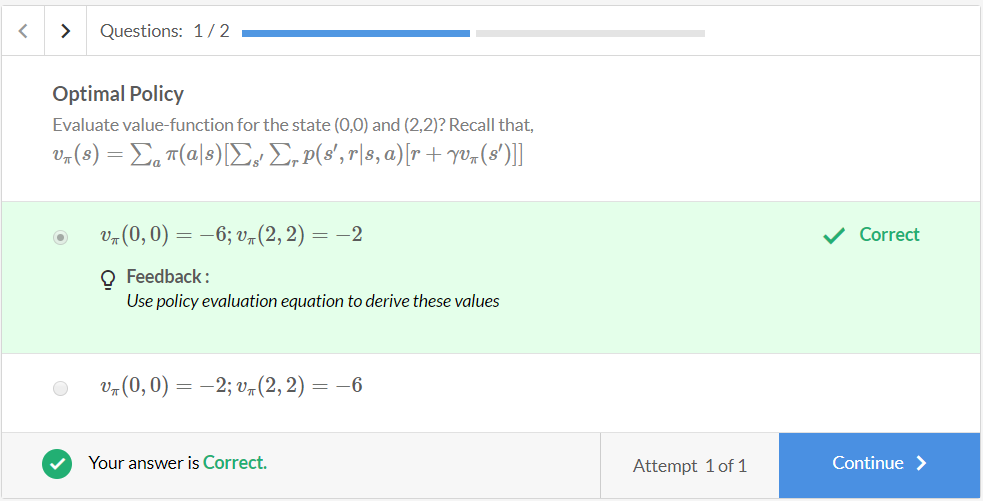


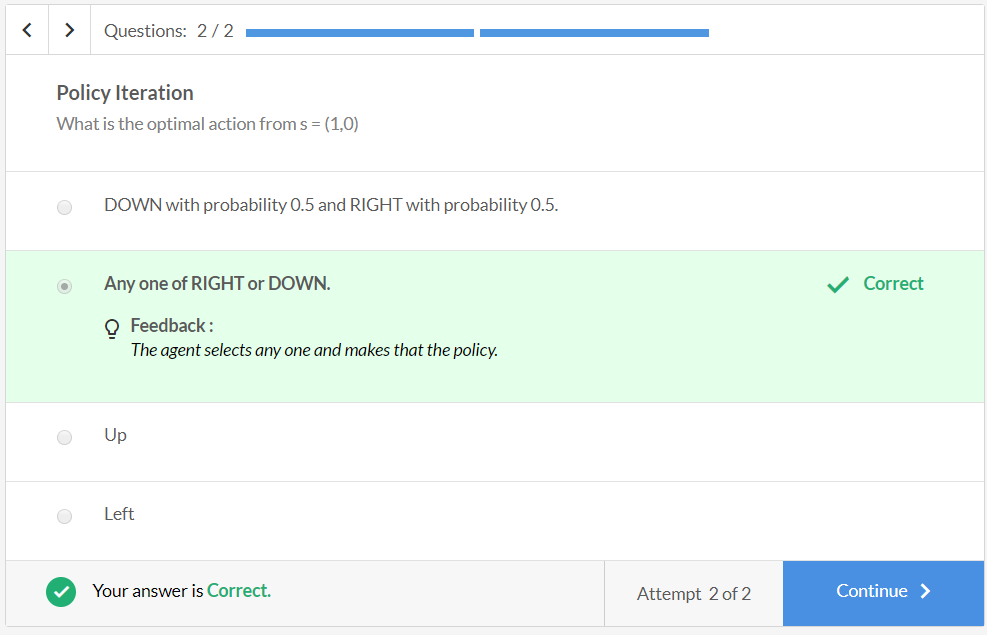
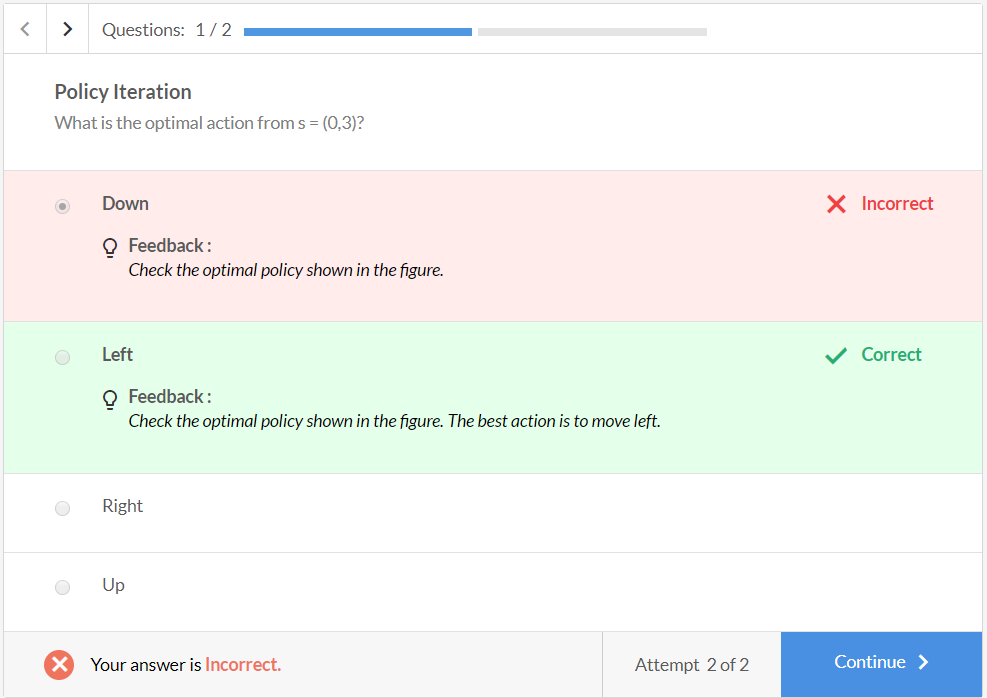




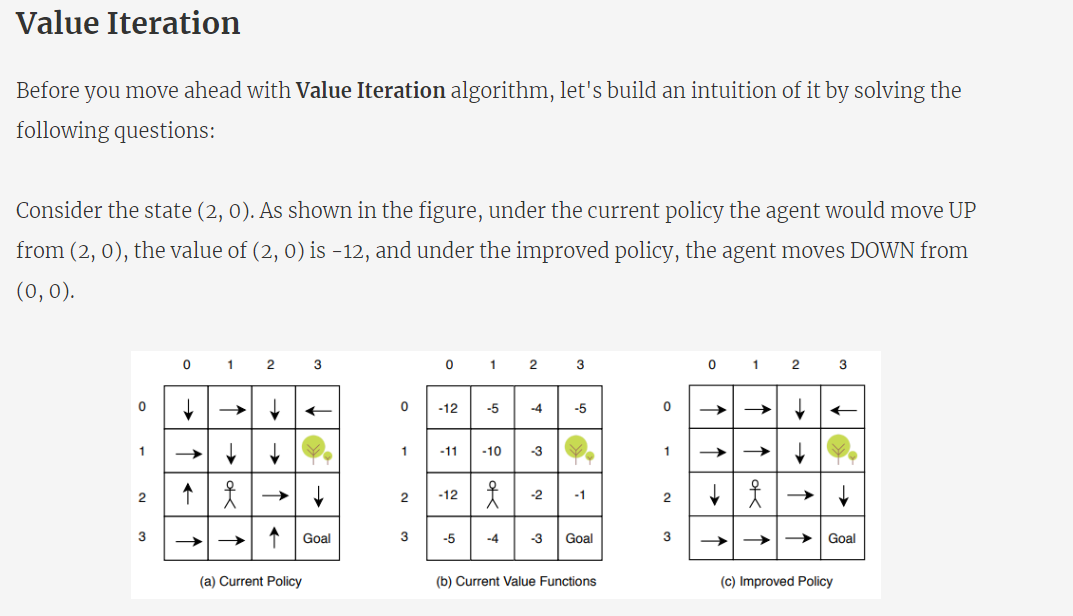


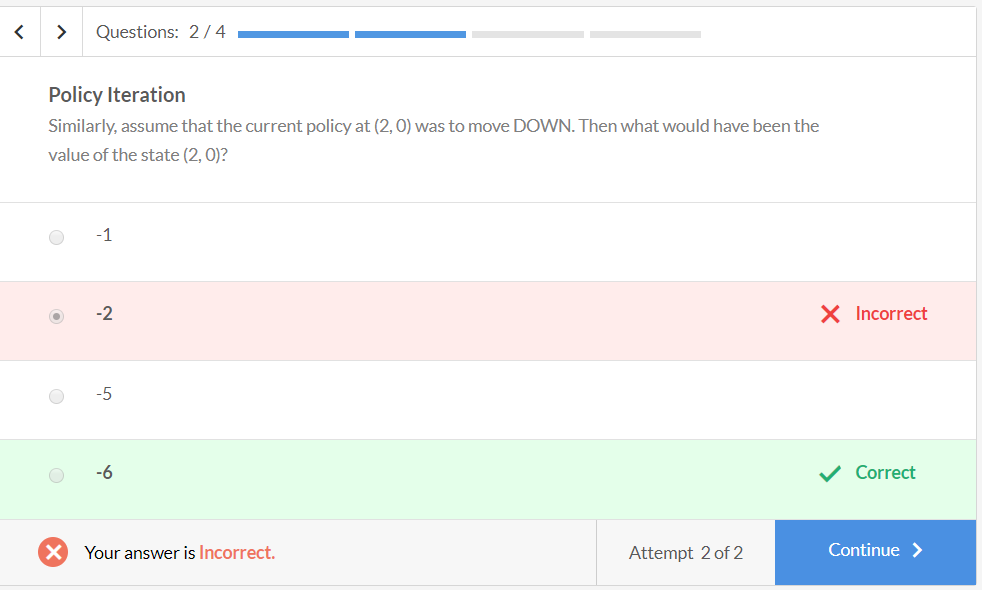
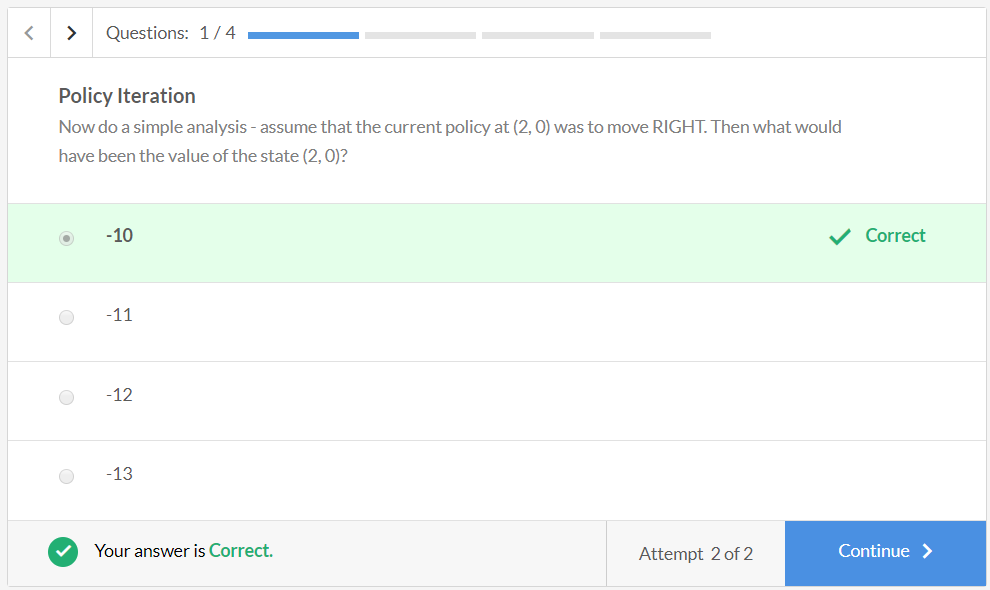


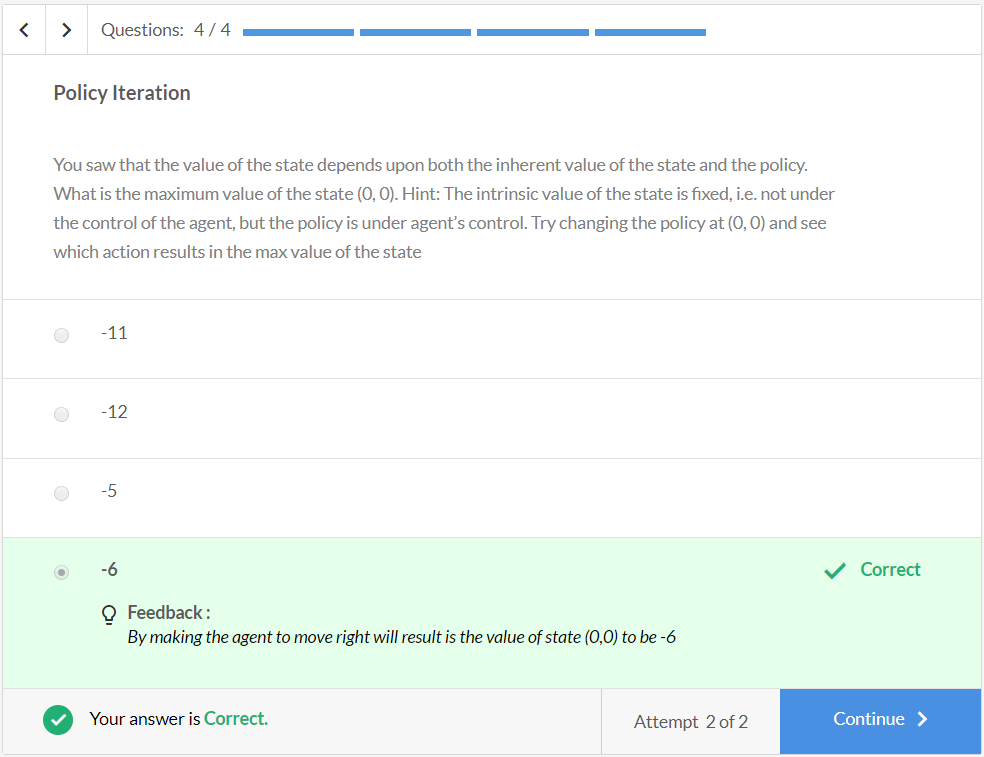
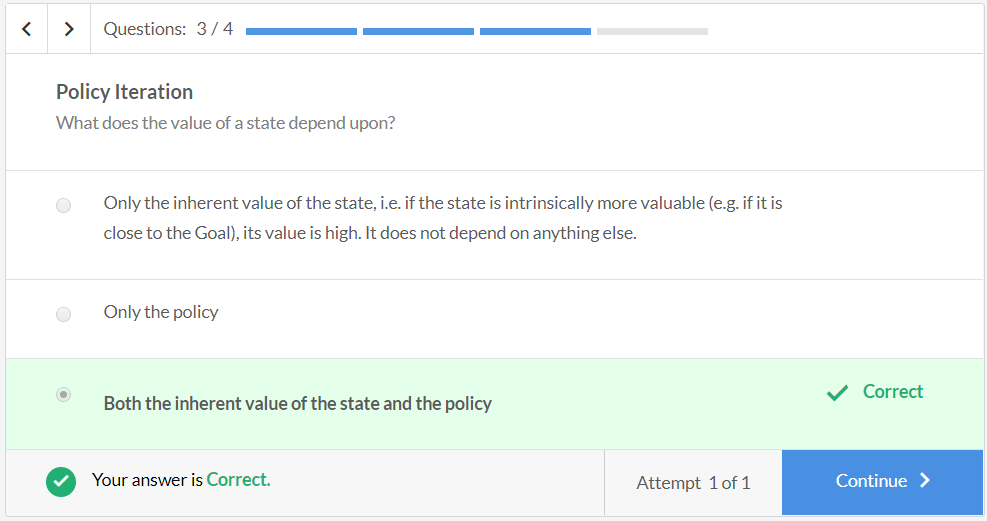


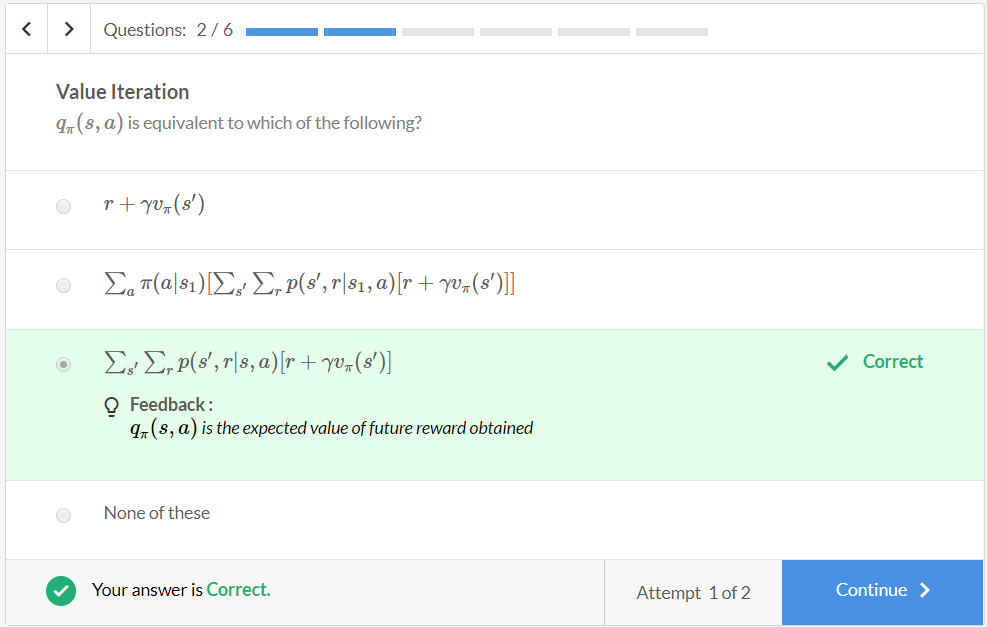
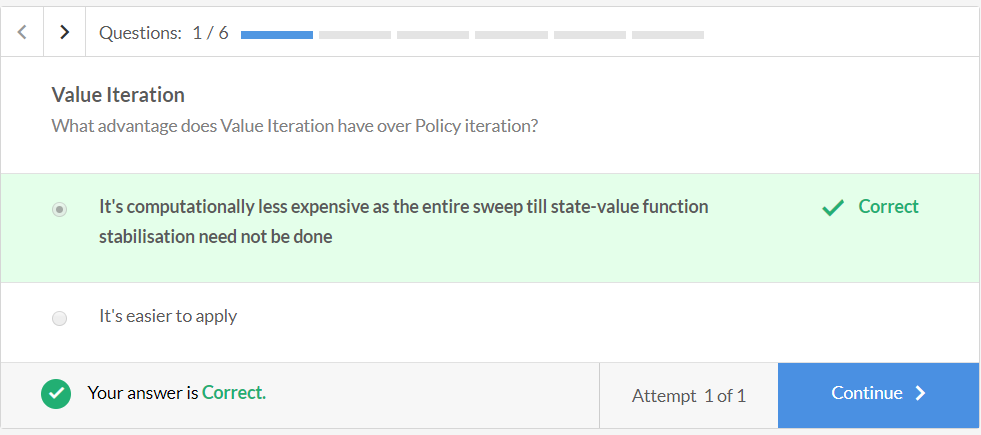


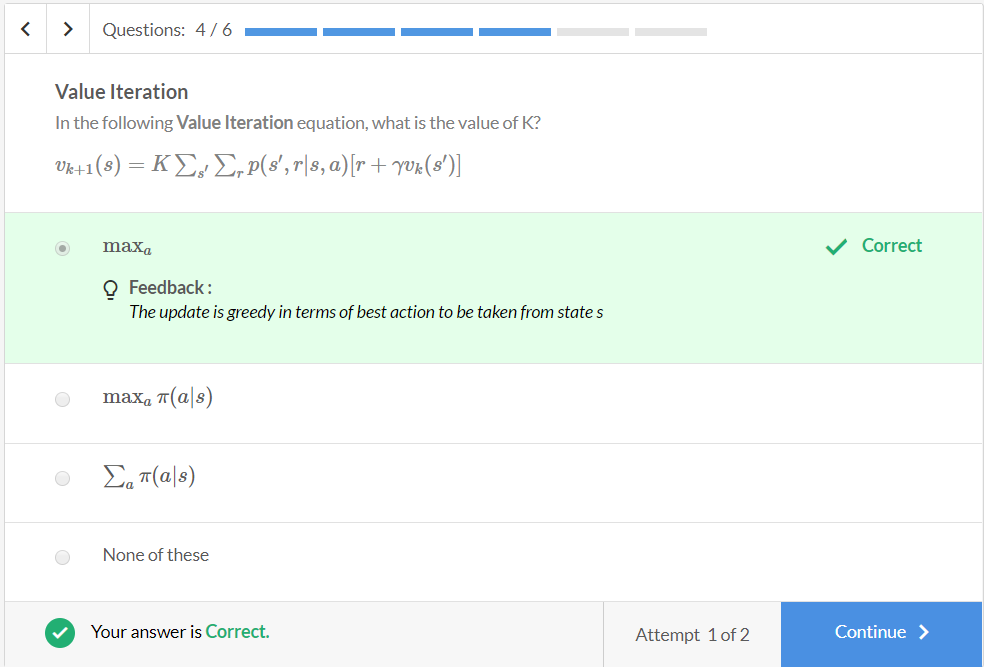
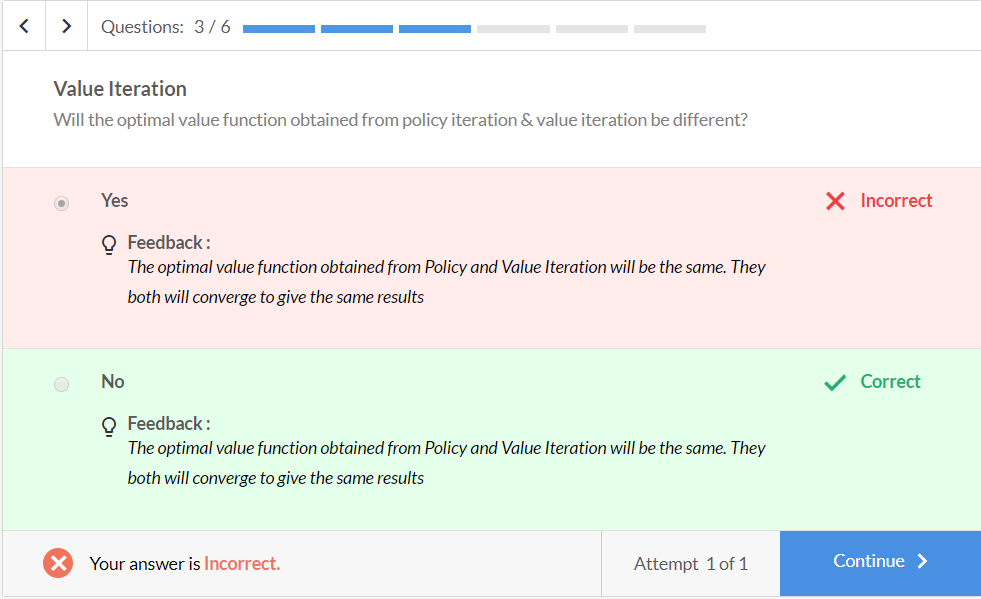
**Value Iteration**

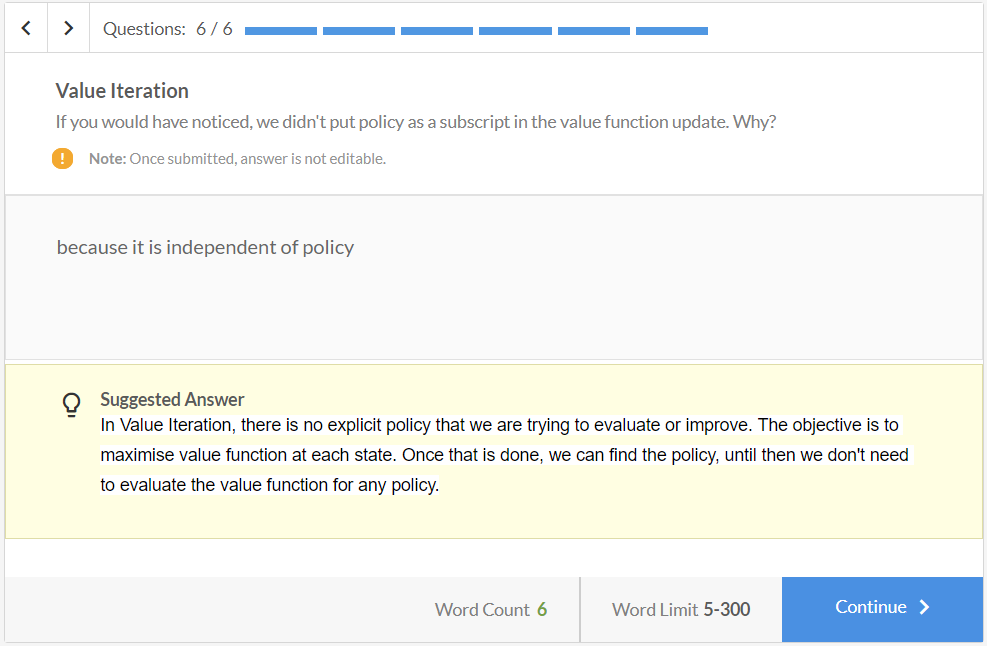
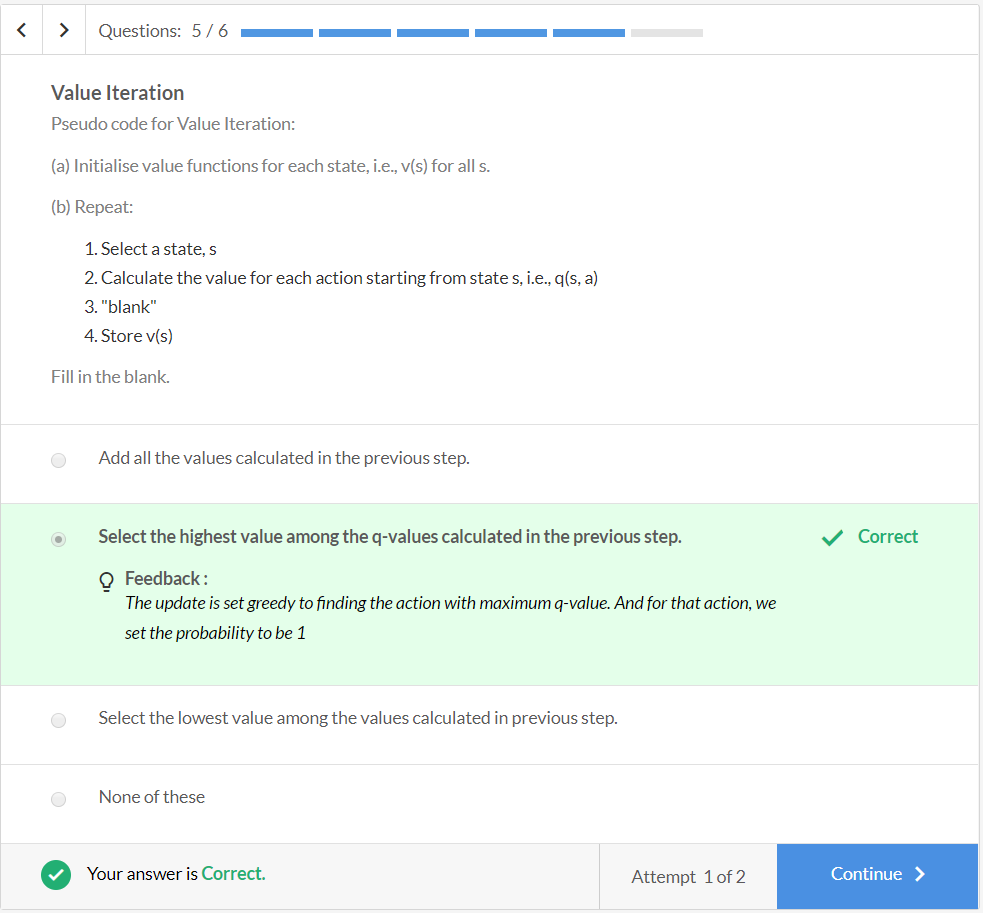


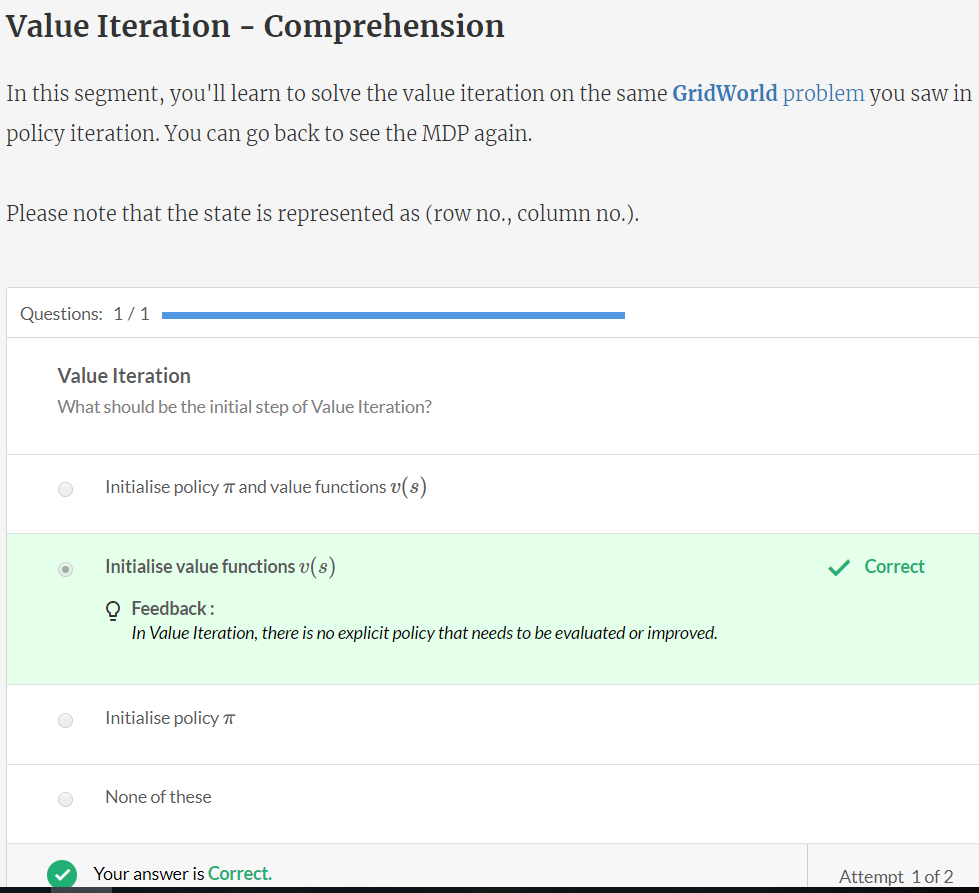


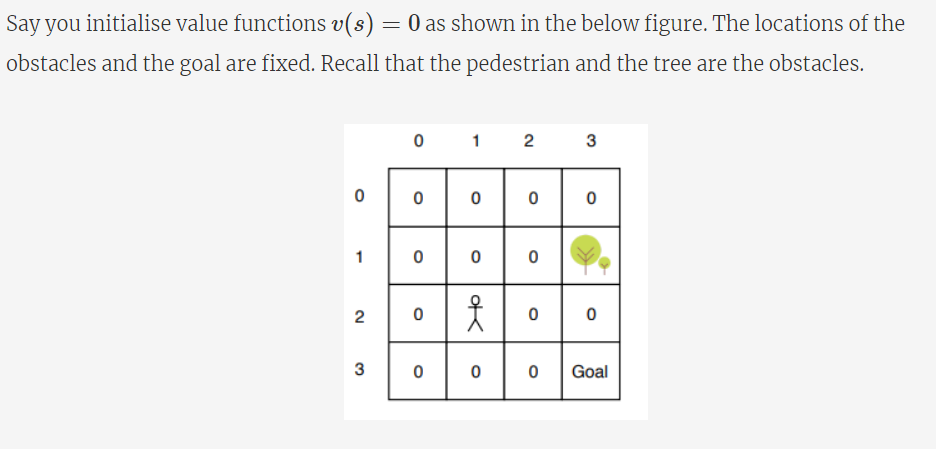


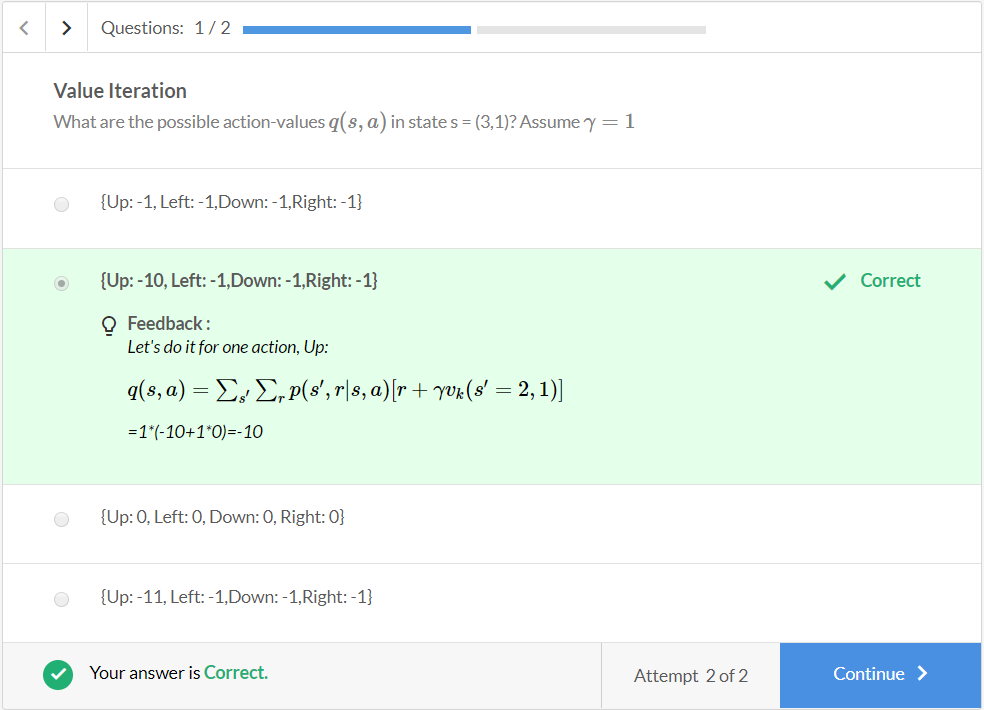
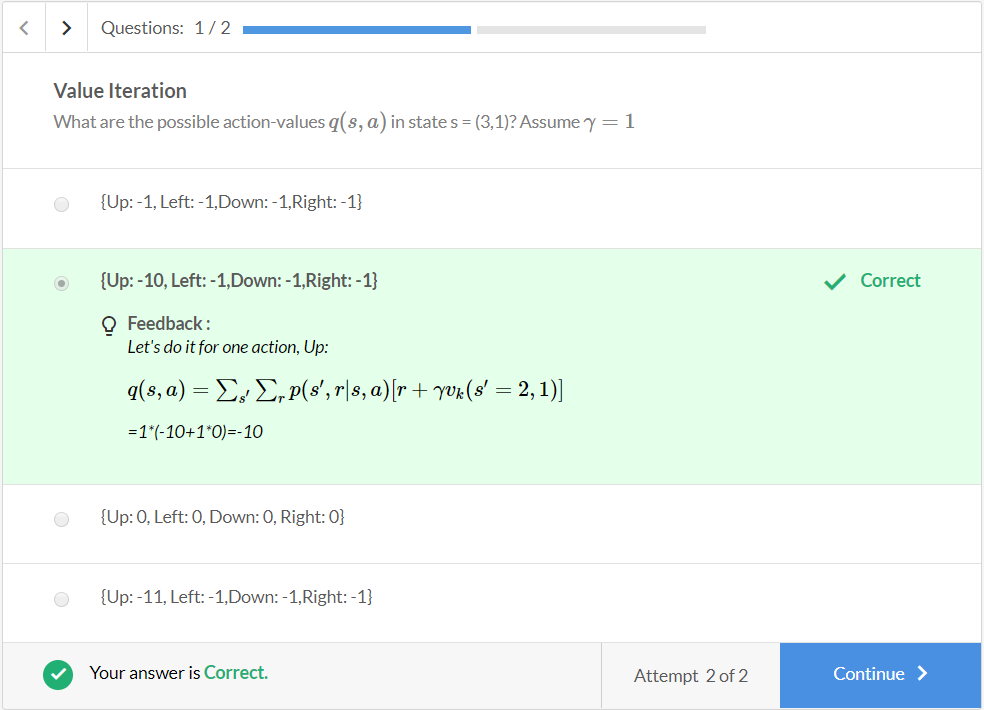


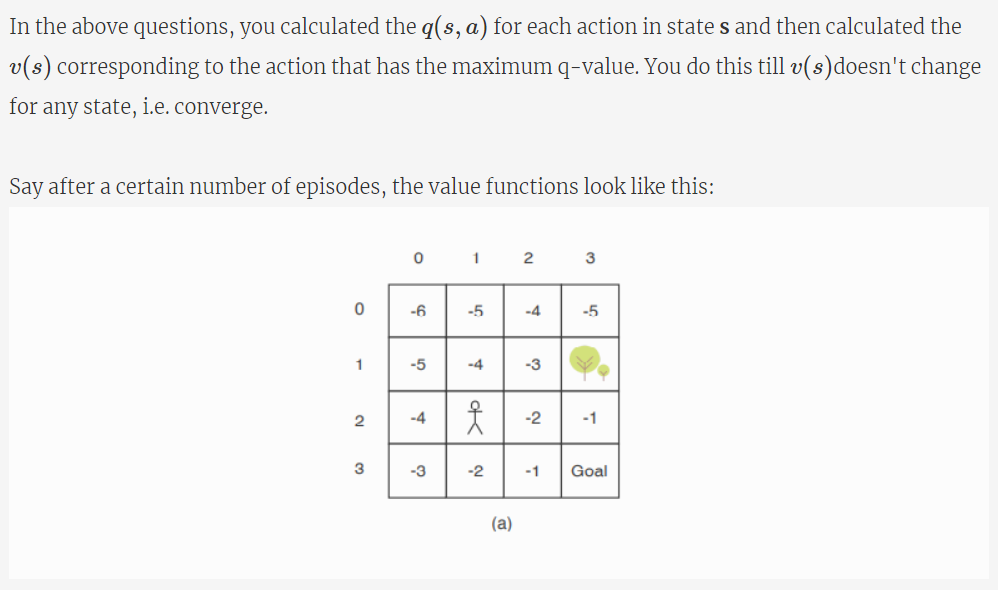


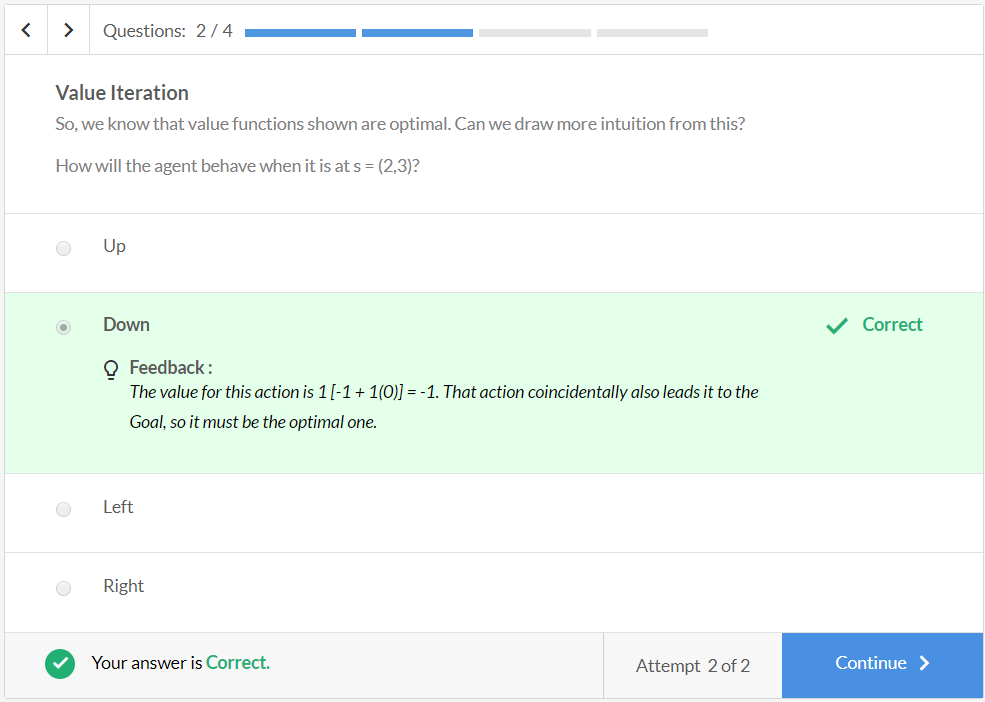
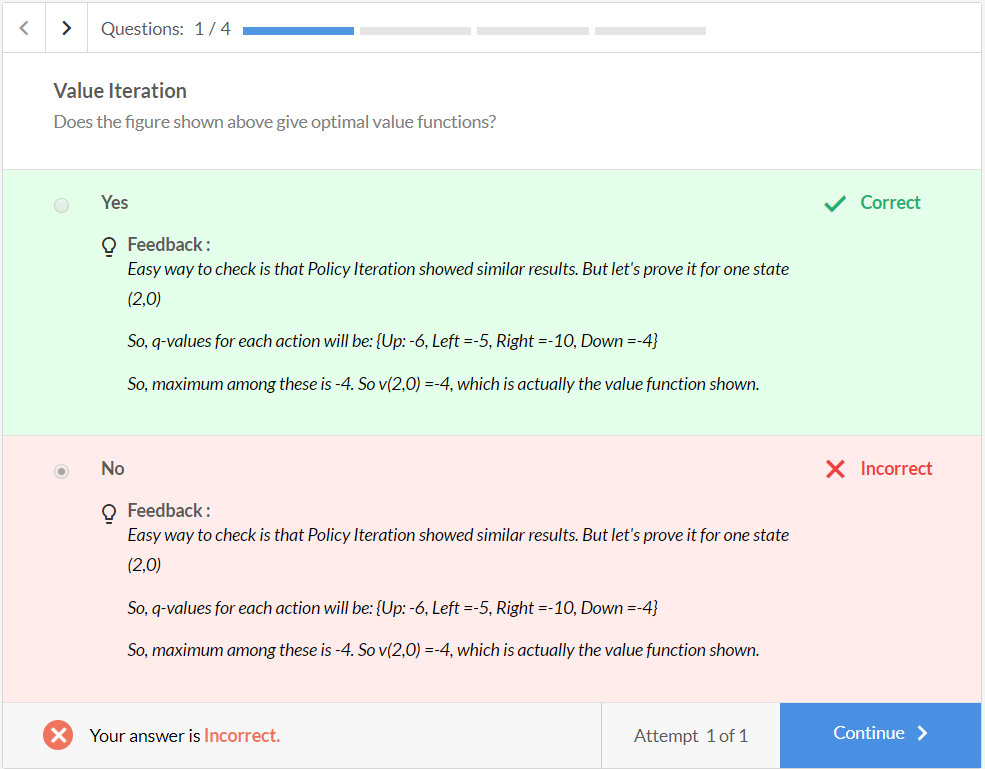


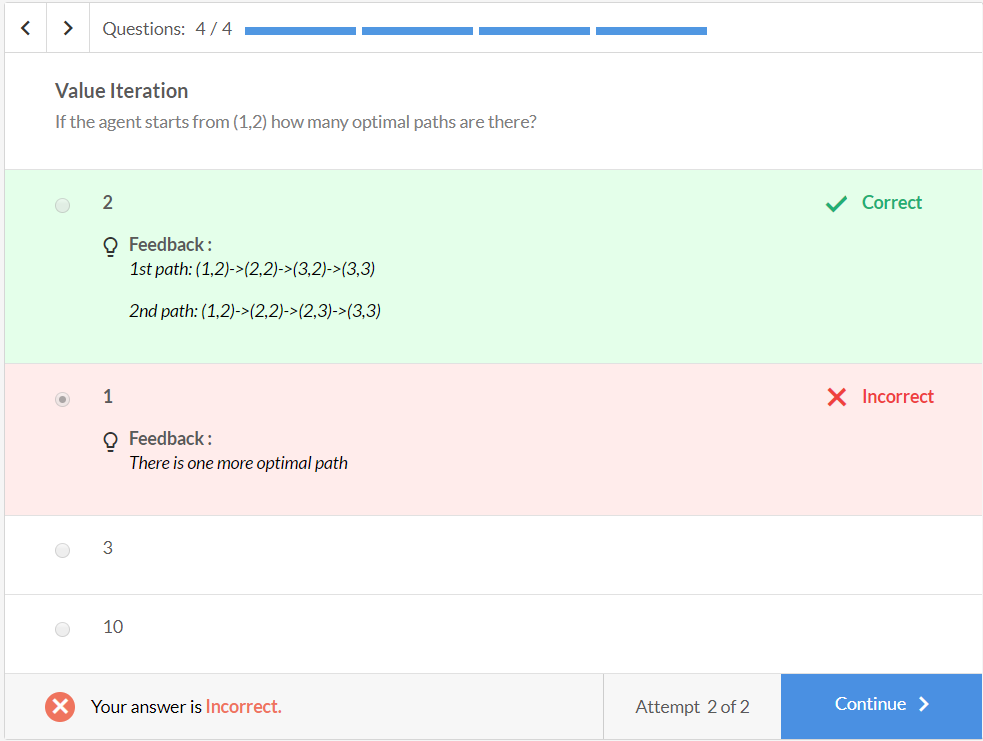
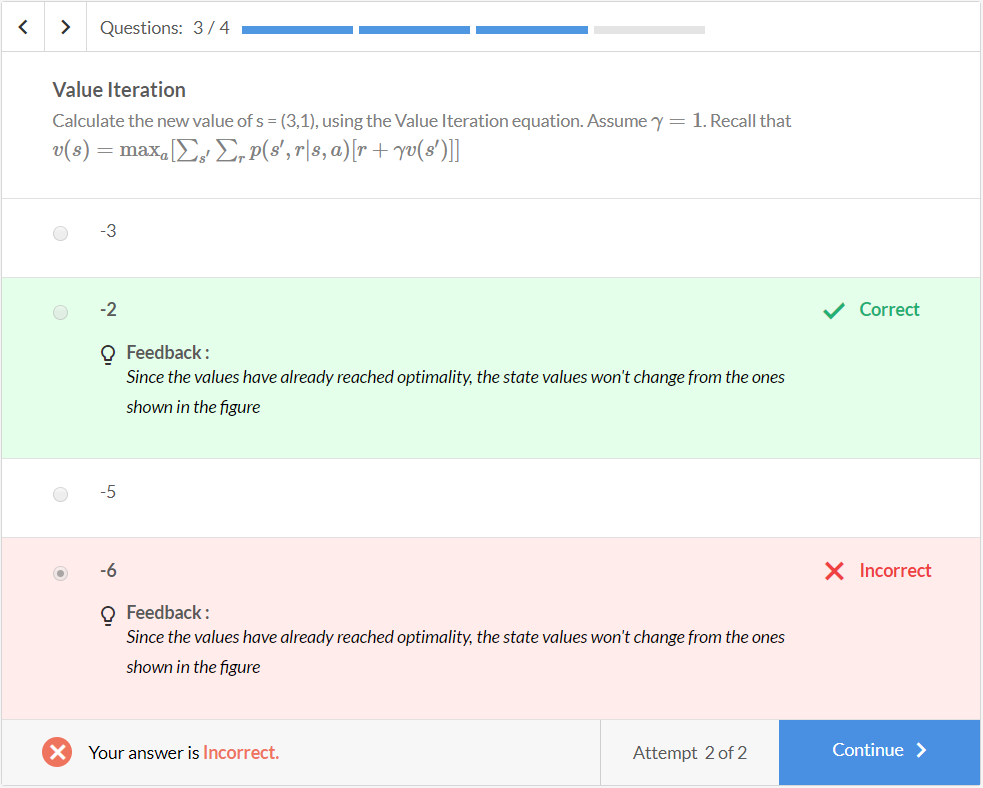












This completes the value iteration.

# Generalised Policy Iteration (GPI)

