Welcome to the second entry in the Unity AI Blog series! For this post, I want to pick up where we left off last time, and talk about how to take a [Contextual Bandit problem](https://blogs.unity3d.com/2017/06/26/unity-ai-themed-blog-entries/), and extend it into a full [Reinforcement Learning](https://en.wikipedia.org/wiki/Reinforcement_learning) problem. In the process, we will demonstrate how to use an agent which acts via a learned Q-function that estimates the long-term value of taking certain actions in certain circumstances. For this example we will only use a simple grid world, and a tabular Q-representation. Fortunately, this, basic idea applies to almost all games. If you like to try out the Q-learning demo, follow the link [here](http://awjuliani.github.io/GridGL/). For a deeper walkthrough of how Q-learning works, continue to the full text below.

The Q-Learning Algorithm

**Contextual Bandit Recap**

The goal when doing Reinforcement Learning is to train an agent which can learn to act in ways that maximizes future expected rewards within a given environment. In the last post in this series, that environment was relatively static. The state of the environment was simply which of the three possible rooms the agent was in, and the actions were choosing which chest within that room to open. Our algorithm learned the Q-function for each of these state-action pairs: Q(s, a). This Q-function corresponded to the expected future reward that would be acquired by taking that action within that state over time. We called this problem the “Contextual Bandit.”

**The Reinforcement Learning Problem**

The lack of two things kept that Contextual Bandit example from being a proper Reinforcement Learning problem: *sparse rewards*, and *state transitions*. By sparse rewards, we refer to the fact that the agent does not receive a reward for every action it takes. Sometimes these rewards are “delayed,” in that certain actions which may in fact be optimal, may not provide a payout until a series of optimal actions have been taken. To use a more concrete example, an agent may be following the correct path, but it will only receive a reward at the end of the path, not for every step along the way. Each of those actions may have been essential to getting the final reward, even if they didn’t provide a reward at the time. We need a way to perform *credit assignment*, that is, allowing the agent to learn that earlier actions were valuable, even if only indirectly.

The second missing element is that in full reinforcement learning problems there are transitions between states. This way, our actions not only produce rewards according to a reward function: *R(s, a) ⇨  r*, but also produce new states, according to a state transition function: *P(s, a) ⇨ s’*. A concrete example here is that every step taken while walking along a path brings the agent to a new place in that path, hence a new state.Therefore we want our agent not only to learn to act to optimize the current possible reward, but act to move toward states we know provide even larger rewards.

**Bellman Updates**

While these two added elements of complexity may at first seem unrelated, they are in fact directly connected. Both imply a relationship between future states our agent might end up in, and future rewards our agent might receive. We can take advantage of this relationship to learn to take optimal actions under these circumstances with a simple insight. Namely, that under a “true” optimal Q-function (a theoretical one which we may or may not ever reach ourselves) the value of a current state and action can be decomposed into to the immediate reward r plus the discounted maximum future expected reward from the next state the agent will end up in for taking that action:

[https://blogs.unity3d.com/wp-content/uploads/2017/08/Reinf_Learning_image4.png](https://blogs.unity3d.com/wp-content/uploads/2017/08/Reinf_Learning_image4.png)

This is called the Bellman equation, and can be written as follows:

[https://blogs.unity3d.com/wp-content/uploads/2017/08/Reinf_Learning_image5.png](https://blogs.unity3d.com/wp-content/uploads/2017/08/Reinf_Learning_image5.png)

Here 𝛄 (gamma) is a discount term, which relates to how much we want our agent to care about future possible rewards. If we set 𝛄 to 1.0, our agent would value all possible future rewards equally, and in training episodes which never end, the value estimate might increase to infinity. For this reason, we set 𝛄 to something greater than 0 and less than 1. Typical values are between 0.7 and 0.99.

The Bellman equation is useful because it provides a way for us to think about updating our Q-function by bootstrapping from the Q-function itself. Q\*(s, a) refers to an optimal Q-function, but even our current, sub-optimal Q value estimates of the next state can help push our estimates of the current state in a more accurate direction. Since we are relying primarily on the true rewards at each step, we can trust that the Q-value estimates themselves will slowly improve. We can use the Bellman equation to inform the following new Q-learning update:

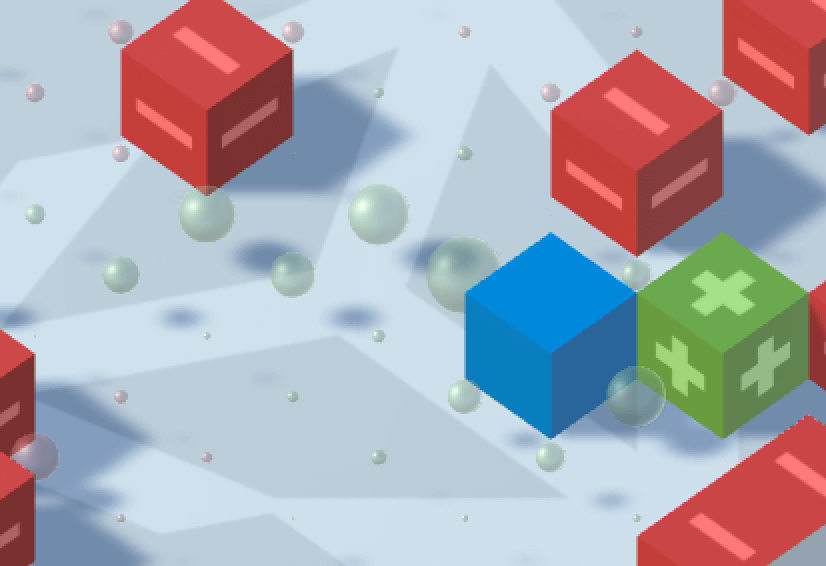
[https://blogs.unity3d.com/wp-content/uploads/2017/08/Reinf_Learning_image6.png](https://blogs.unity3d.com/wp-content/uploads/2017/08/Reinf_Learning_image6.png)

This looks similar to our previous contextual bandit update algorithm, except that our Q-target now includes the discounted future expected reward at the next step.

**Exploration**

In order to ensure that our agent properly explores the state space, we will utilize a form of exploration called *epsilon-greedy*. To use epsilon-greedy, we simply set an epsilon value ϵ to 1.0, and decrease it by a small amount every time the agent takes an action. When the agent chooses an action, it either picks *argmax(Q(s, a))*, the greedy action, or takes a random action with probability ϵ. The intuition is that at the beginning of training our agent’s Q-value estimates are likely to be very poor, but as we learn about the world, and ϵ decreases, our Q-function will slowly correspond more to the true Q-function of the environment, and the actions we take using it will be increasingly accurate.

The Unity Gridworld

[](https://blogs.unity3d.com/wp-content/uploads/2017/08/Reinf_Learning_image12.png)

The blue block corresponds to the agent, the red blocks to the obstacles, and the green block to the goal position. The green and red spheres correspond to the value estimates for each of the states within the gridworld.

To demonstrate a Q-learning agent, we have built a simple gridworld environment using Unity. The environment consists of the following: 1- an agent placed randomly within the world, 2- a randomly placed goal location that we want our agent to learn to move toward, 3- and randomly placed obstacles that we want our agent to learn to avoid. The state (s) of the environment will be an integer which corresponds to the position on the grid. The four actions (a) will consist of (Up, Down, Left, and Right), and the rewards (r) will be: +1 for moving to the state with the goal, -1 for moving to the state with an obstacle, and -0.05 for each step, to encourage quick movement to the goal on the part of the agent. Each episode will end after 100 steps, or when the agent moves to a state with either a goal or obstacle. Like in the previous tutorial, the agent’s Q values will be stored using a table, where the rows correspond to the state, and the columns to the possible actions. You can play with this environment and agent within your Web browser [here](http://awjuliani.github.io/GridGL/), and download the Unity project to modify for use in your own games [here](https://github.com/Unity-Technologies/Q-GridWorld). As the agent explores the environment, colored orbs will appear in each of the gridworld states. These correspond to the agent’s average Q-value estimate for that state. Once the agent learns an optimal policy, it will be visible as a direct value gradient from the start position to the goal.

Going Forward

The agent and environment presented here represent a classic tabular formulation of the Q-learning problem. If you are thinking that perhaps there is not much in common with this basic environment and the ones you find in contemporary games, do not worry. In the years since the algorithm’s introduction in the 90s, there have been a number of important developments to allow Q-learning to be used in more varied and dynamic situations. One prime example is deepmind’s Deep Q-Network which was used to learn to play dozens of different ATARI games directly from pixels, a feat impossible using only a lookup table like the one here. In order to accomplish this, they utilized an agent which was controlled by a Deep Neural Network (DNN). By using a neural network it is possible to learn a generalized Q-function which can be applied to completely unseen states, such as novel combinations of pixels on a screen.

In the next few weeks we will release an interface with a set of algorithms and example projects to allow for the training of similar Deep Reinforcement Learning agent in Unity games and simulations. For a sneak peek of what these tools are capable of, you can check out the video link [here](https://www.youtube.com/watch?v=fiQsmdwEGT8&t=2s). While this initial release will be limited, and aimed primarily at those working in research, industry, and game QA testing, we at Unity are excited about the possibilities opened up by using modern Deep Learning methods to learn game behaviour. We hope that as this work matures, it will spark interest in using ML within game to control complex NPC behavior, game dynamics, and more. We are at the very beginning of exploring using Deep Learning in games, and we look forward to you continuing with us on this journey.

[Unity AI-themed Blog Entries](https://blogs.unity3d.com/kr/2017/06/26/unity-ai-themed-blog-entries/)

[**Arthur Juliani**](https://blogs.unity3d.com/kr/author/arthur-juliani/)   
6월 26, 2017•[22](https://blogs.unity3d.com/kr/2017/06/26/unity-ai-themed-blog-entries/#com)

[Tutorial: Behave 2 for Unity](https://blogs.unity3d.com/kr/2014/04/28/tutorial-behave-2-for-unity/)

[**Kamio**](https://blogs.unity3d.com/kr/author/kameo/)   
4월 28, 2014•[7](https://blogs.unity3d.com/kr/2014/04/28/tutorial-behave-2-for-unity/#com)

[Asset Store](https://blogs.unity3d.com/kr/category/asset-store/)

[유니티 머신러닝 에이전트 소개](https://blogs.unity3d.com/kr/2017/09/19/introducing-unity-machine-learning-agents/)

[**Arthur Juliani**](https://blogs.unity3d.com/kr/author/arthur-juliani/)   
9월 19, 2017•[72](https://blogs.unity3d.com/kr/2017/09/19/introducing-unity-machine-learning-agents/#com)

[Introducing Unity Labs’ New Global Research Fellowship Program](https://blogs.unity3d.com/kr/2017/06/27/introducing-unity-labs-new-research-fellowship-program/)

[**Diana Ford**](https://blogs.unity3d.com/kr/author/dianaf/)   
6월 27, 2017•[12](https://blogs.unity3d.com/kr/2017/06/27/introducing-unity-labs-new-research-fellowship-program/#com)

[커뮤니티](https://blogs.unity3d.com/kr/category/community-news/) [이벤트](https://blogs.unity3d.com/kr/category/events/) [테크놀로지](https://blogs.unity3d.com/kr/category/technology/)

[Unity AI-themed Blog Entries](https://blogs.unity3d.com/kr/2017/06/26/unity-ai-themed-blog-entries/)

[**Arthur Juliani**](https://blogs.unity3d.com/kr/author/arthur-juliani/)   
6월 26, 2017•[22](https://blogs.unity3d.com/kr/2017/06/26/unity-ai-themed-blog-entries/#com)

[Tutorial: Behave 2 for Unity](https://blogs.unity3d.com/kr/2014/04/28/tutorial-behave-2-for-unity/)

[**Kamio**](https://blogs.unity3d.com/kr/author/kameo/)   
4월 28, 2014•[7](https://blogs.unity3d.com/kr/2014/04/28/tutorial-behave-2-for-unity/#com)

[Asset Store](https://blogs.unity3d.com/kr/category/asset-store/)

[유니티 머신러닝 에이전트 소개](https://blogs.unity3d.com/kr/2017/09/19/introducing-unity-machine-learning-agents/)

[**Arthur Juliani**](https://blogs.unity3d.com/kr/author/arthur-juliani/)   
9월 19, 2017•[72](https://blogs.unity3d.com/kr/2017/09/19/introducing-unity-machine-learning-agents/#com)

[Introducing Unity Labs’ New Global Research Fellowship Program](https://blogs.unity3d.com/kr/2017/06/27/introducing-unity-labs-new-research-fellowship-program/)

[**Diana Ford**](https://blogs.unity3d.com/kr/author/dianaf/)   
6월 27, 2017•[12](https://blogs.unity3d.com/kr/2017/06/27/introducing-unity-labs-new-research-fellowship-program/#com)

[커뮤니티](https://blogs.unity3d.com/kr/category/community-news/) [이벤트](https://blogs.unity3d.com/kr/category/events/) [테크놀로지](https://blogs.unity3d.com/kr/category/technology/)

* 1
* 2
* 3
* 4

15 코멘트

[코멘트 구독](https://blogs.unity3d.com/kr/2017/08/22/unity-ai-reinforcement-learning-with-q-learning/feed/)

코멘트를 달 수 없습니다.

1. https://secure.gravatar.com/avatar/5a252fe1f8042c7da7becfecd3193f4a?s=60&d=mm&r=pg

Fredrik Chrislock

[9월 13, 2017 10:27 오후](https://blogs.unity3d.com/kr/2017/08/22/unity-ai-reinforcement-learning-with-q-learning/#comment-361775)

Great initiative! I look forward to more posts in the future.

Are there any plans to integrate support for deep learning toolkits? I know the CNTK team is working on a C# wrapper. This could be a great possibility for Unity to be the first game engine to support next gen AI!