Unity Machine Learning

Penguin Simulation

Bao Nguyen

Table of Contents

I. Overview

1. Unity Machine Learning Agents Toolkit

"The Unity Machine Learning Agents Toolkit (ML-Agents Toolkit) is an open-source Unity plugin that enables games and simulations to serve as environments for training intelligent agents. Agents can be trained using reinforcement learning, imitation learning, neuro-evolution, or other machine learning methods through a simple-to-use Python API."

Hence, basically, this toolkit is kind of a plugin that helps training agents for machine learning in Unity. However, it is still in beta version, so there could be some or many errors while manually setting up and running it from the console or editor.

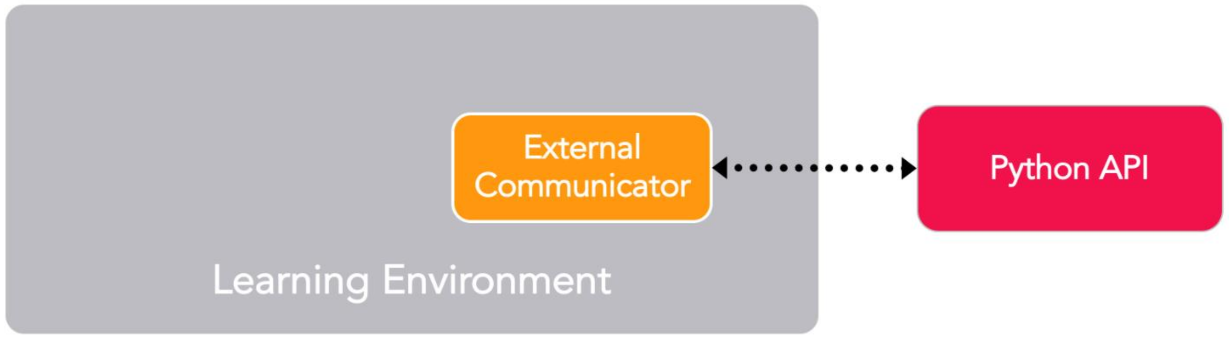
Different version also has some changes in code and function. Check Migrating document for more information.

Thus, some part of this document may get depreciated or changed.

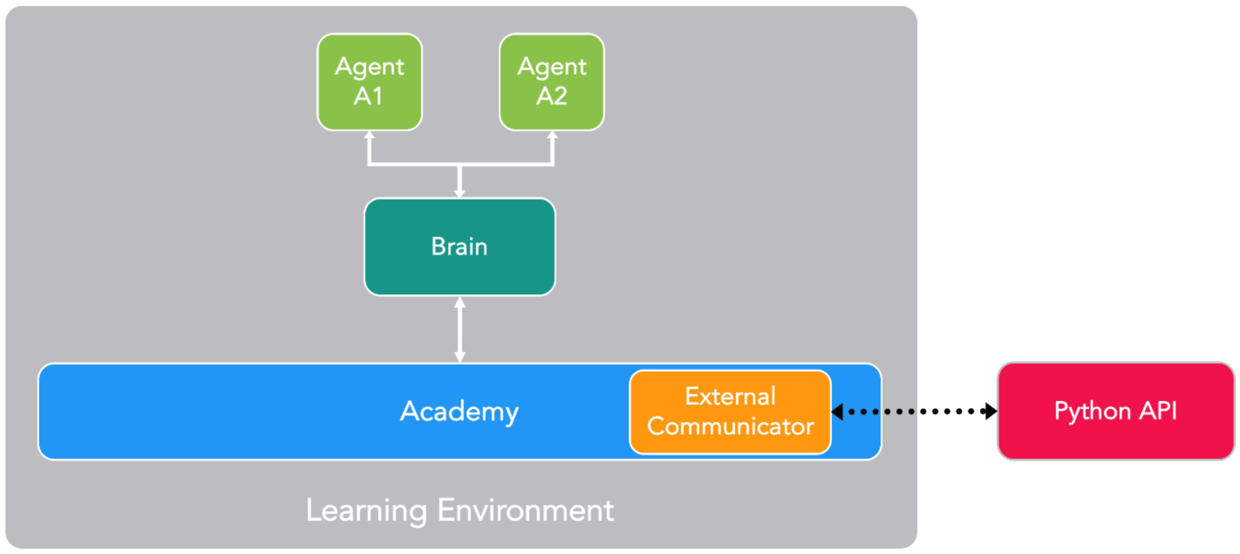
2. Key Components

Three high-level components are included:

- **Learning Environment**: contain Unity scene and all game characters.
- **External Communicator**: lives in Learning Environment, which connects the Learning Environment with Python API to communicate each other.
- **Python API**: not part of Unity, contains all the machine learning algorithms that are used for training (learning a behavior or policy).



Furthermore, Learning Environment also includes an important additional component – **Agent**. An Agent is a character in a scene which generates Observations, and passes them to its Policy. Then the Policy makes Decisions for it to perform the Actions and then assigns Rewards (both positive and negative) for each action.



To summary, "Each character is attached to an Agent, and each Agent has a Policy. The Policy receives observations and rewards from the Agent and returns actions. The Academy ensures that all the Agents are in sync in addition to controlling environment-wide settings."

3. Installation

For this Machine Learning simulation, I use the latest ML-Agents Toolkit 0.15.1 from the Unity ML-Agents GitHub and install that package in an 3D Template Project.

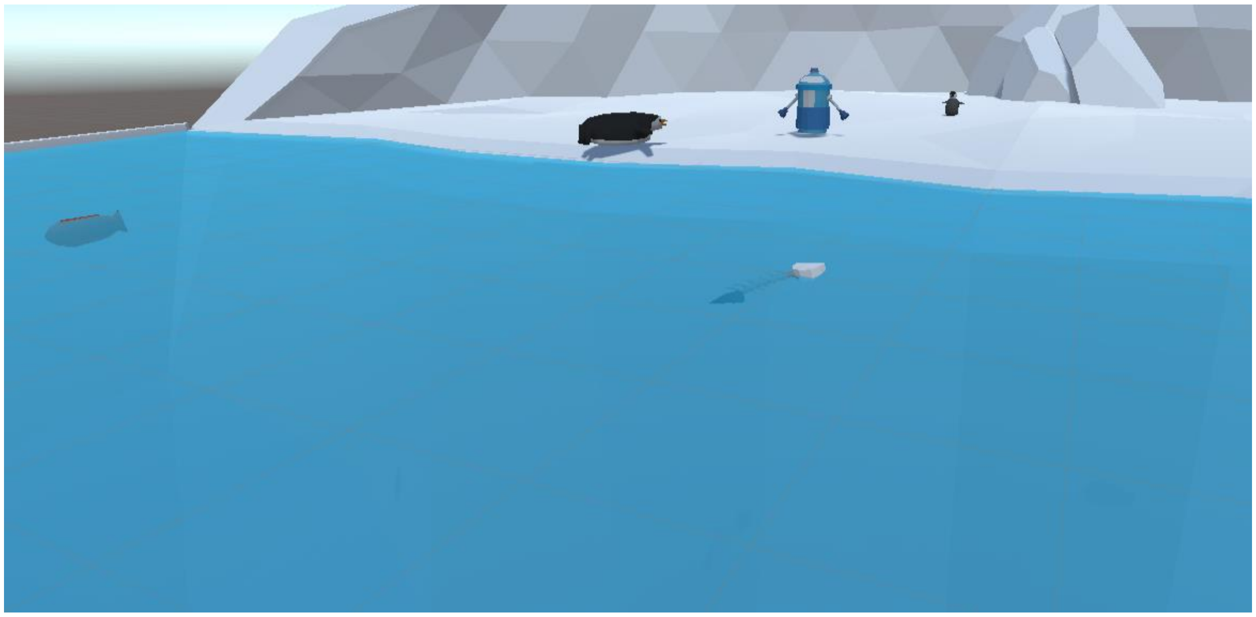
A Python Virtual Environment is needed for ML-Agents as well. In this section, it is recommended to install Python 3.7 through Anaconda because it is easier to manage multiple Python environments. After activating the environment, you can install ml-agents-env and ml-agents Python libraries.

Otherwise, you could follow the Installation document of Unity ml-agents on GitHub, although I find it hard to follow properly.

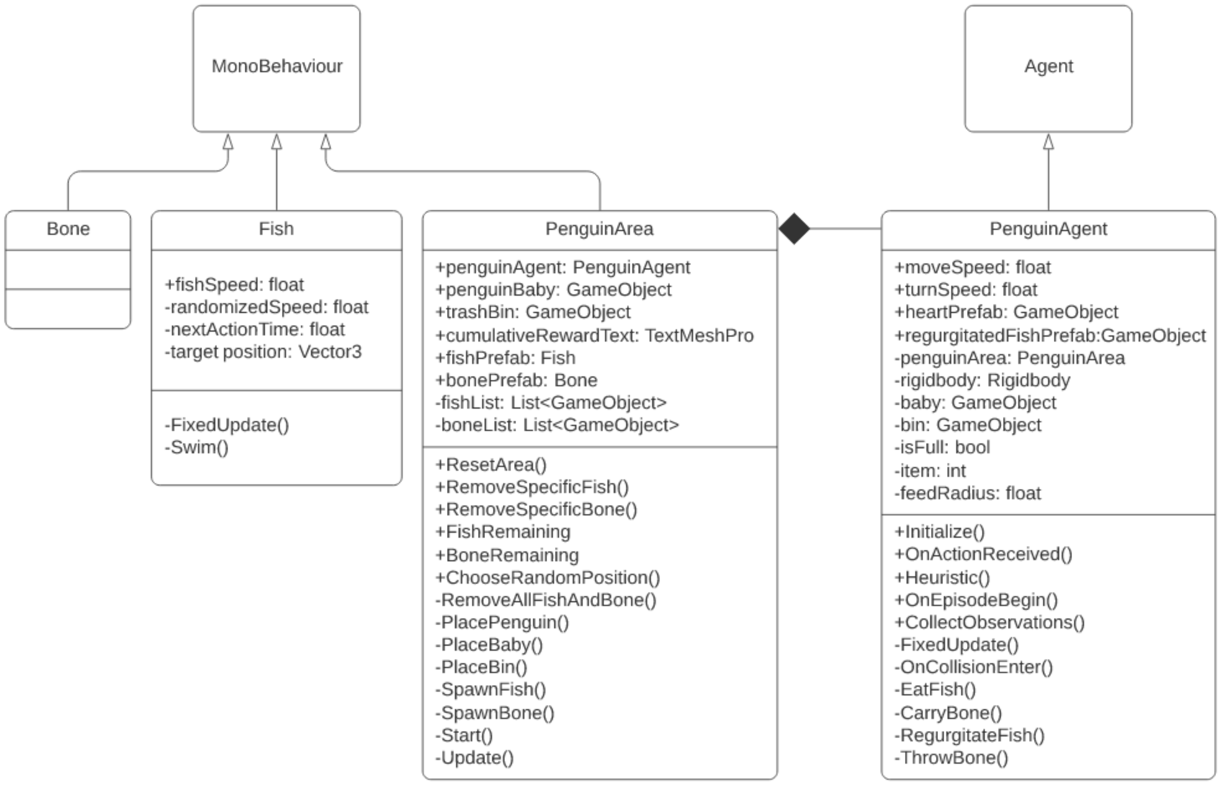
II. Real practice

For easier understanding, I will try to explain Unity machine learning aspects along with a real simulation. The simulation utilizes reinforcement learning method.

I made a simulation following a Penguin tutorial, where a penguin tries to swim to catch the fish, then get back to lane and feed its baby. In our more advanced version, the penguin must learn to differentiate the fish and the bones, and proper places to bring them to. The fish go to the baby penguin, while the bones go to a trash bin nearby.



There are total 4 classes in this simulation. While the fish, bone and area are inherited from the default MonoBehaviour of Unity, the big penguin is inherited from Agent class.

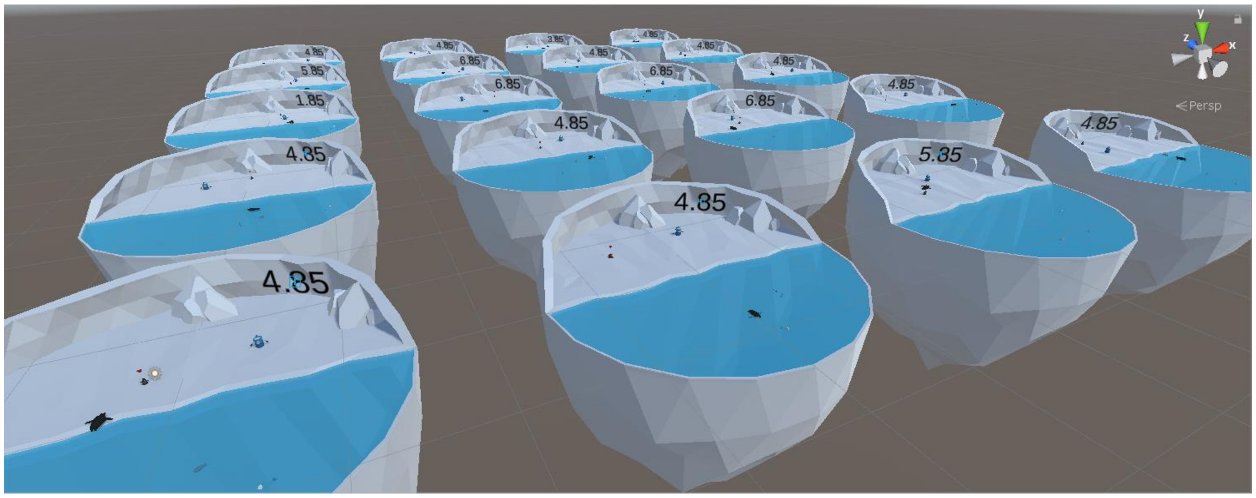


1. PenguinArea

Although inheriting from MonoBehaviour, PenguinArea class is still using MLAgents namespace for creating "curriculum" for the Agent, which is the amount of fish and bone. It also acts like a Learning Environment for the Penguin to observe and learn.

ResetArea() is initially called on Start(), which will spawn penguin, baby, bin, fish, bones at random locations. By default (if no curriculum applied), there are 1 penguin, 1 baby, 1 bin, and 4 fish and 2 bones.

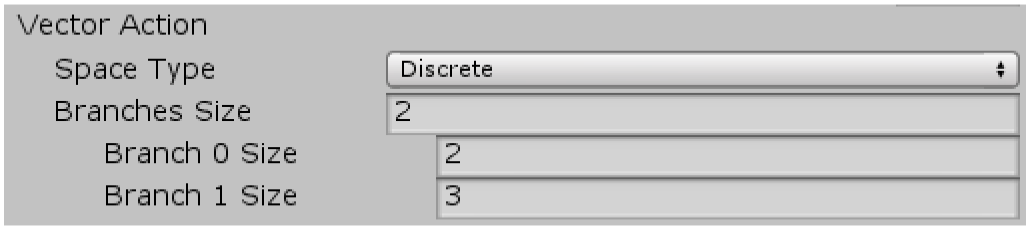
Furthermore, there could be multiple PenguinArea-s for more efficient and faster training.



The number on each area show the cumulative Reward that the Agent in there has gained during one "episode".

2. PenguinAgent

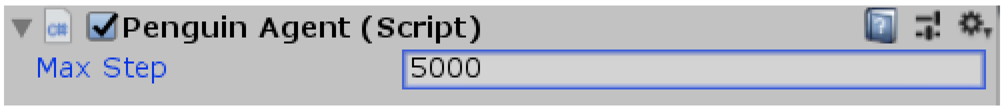
First of all, Initialize() is called for once when the Agent wakes up. Then it receives and responds to commands in OnActionReceived(). These commands may from a neural network or a human layer, but this function treats them the same. Some modification in the inspector (mostly Behavior Parameters) of the prefab are needed too, to match the code.



Vector Action

Space Type	Discrete
Branches Size	2
Branch 0 Size	2
Branch 1 Size	3

In this case, there are 2 vector Action to receive. One is for forward direction (stand still or move forward), and the other is for turning (not turn, turn left, turn right). After that, this function applies the movement and rotation, and then add a tiny negative reward (punishment) in each step (there are 5000 steps per episode). This would push the Agent to complete its task as fast as possible.

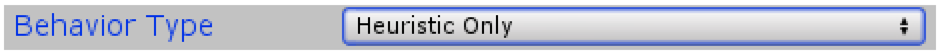


Penguin Agent (Script)

Max Step	5000

Max Steps means the agent will reset automatically after 5,000 steps.

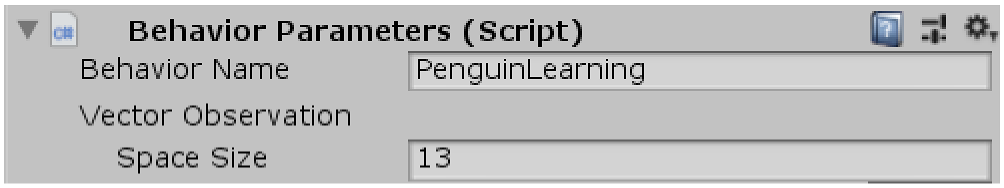
"The Heuristic() function allows control of the agent without a neural network. This function will read inputs from the human player via the keyboard, convert them into actions, and return a list of those actions". This is only applied if you want to control the Agent and set Behavior Type to Heuristic Only. After, you can control the penguin with WASD keys.



Behavior Type	Heuristic Only

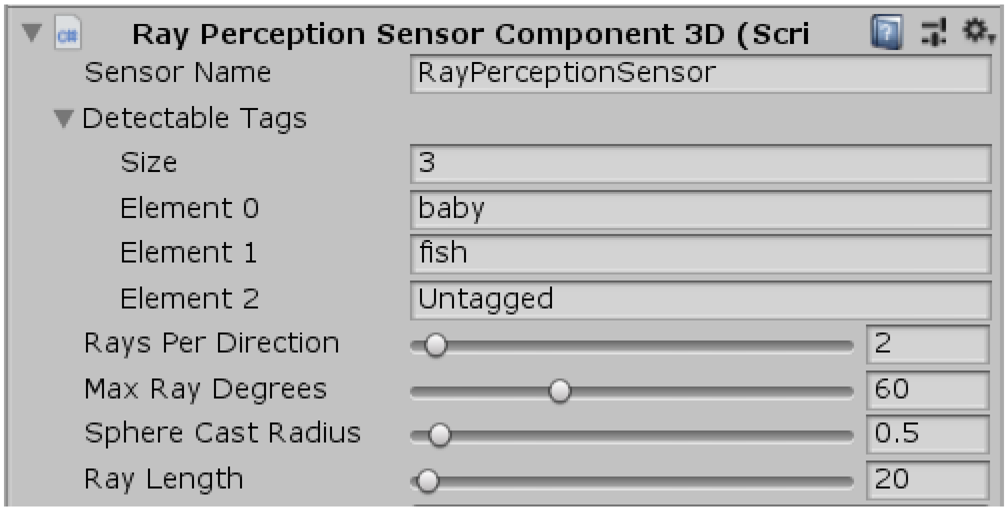
When the penguin clears the lake to empty, OnEpisodeBegin() will be automatically called. It resets the area by using ResetArea() of PenguinArea class, and give a new value for the feed radius for new "curriculum". By default, it is 0, which means that the penguin has to actually touch the baby or the bin to feed it. This is also called when the max step is reached, which is 5000.

The Agent utilizes Vector Sensor to observe the environment. The CollectObservations() method works with Ray Perception Sensor Component 3D to achieve this. In this case, there are 13 values to observe, including 3 Vector3 (3 values each) and 1 int, 1 bool, 2 floats (1 value each). It also uses the rays to check the tags.



Behavior Parameters (Script)

Behavior Name	PenguinLearning
Vector Observation	
Space Size	13



Ray Perception Sensor Component 3D (Scri

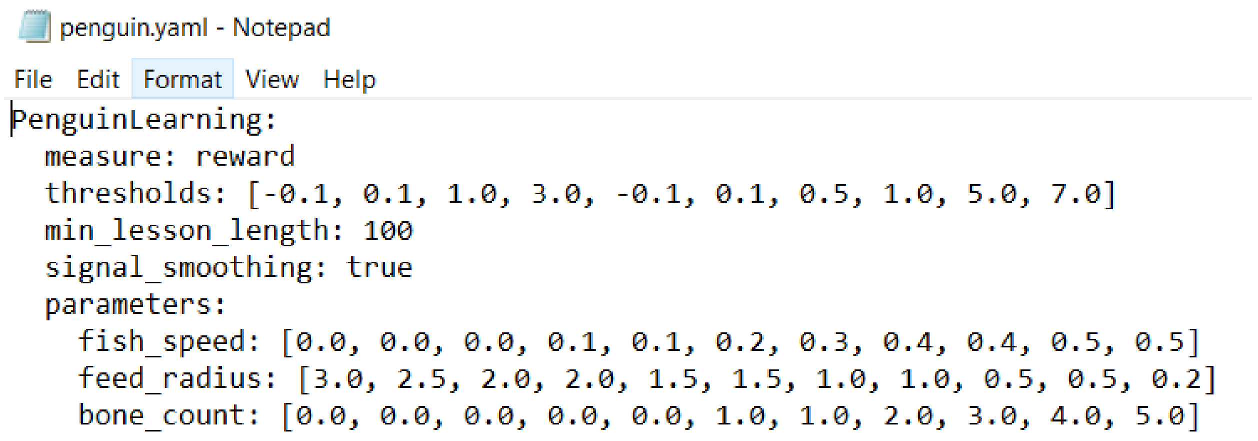
Sensor Name	RayPerceptionSensor
Detectable Tags	
Size	3
Element 0	baby
Element 1	fish
Element 2	Untagged
Rays Per Direction	2
Max Ray Degrees	60
Sphere Cast Radius	0.5
Ray Length	20

In FixedUpdate(), I request a decision every 5 steps. “It is common in reinforcement learning to ask for a decision and take that action a few times in a row before asking for a new decision. It saves on processing and probably reduces jitter”. This function also checks if the Agent is close enough to feed the fish to baby or throw trash to the bin, based on the feed radius.

OnCollisionEnter() simply test to collision with the items with tags and respond accordingly. The Agent gains a reward of 1 whenever it picks up a fish or a bone, and another 1 when it brings back to the right place. However, if it is wrong place, the Agent will receive a punishment of -1.

3. Curriculum Learning

As mentioned above, without curriculum, the penguin Agents would take a lot longer to learn. I am using curriculum to gradually increase the difficulty for our penguins and speed up training significantly.

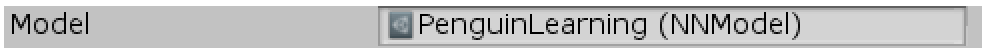


```
PenguinLearning:
  measure: reward
  thresholds: [-0.1, 0.1, 1.0, 3.0, -0.1, 0.1, 0.5, 1.0, 5.0, 7.0]
  min_lesson_length: 100
  signal_smoothing: true
  parameters:
    fish_speed: [0.0, 0.0, 0.0, 0.1, 0.1, 0.2, 0.3, 0.4, 0.4, 0.5, 0.5]
    feed_radius: [3.0, 2.5, 2.0, 2.0, 1.5, 1.5, 1.0, 1.0, 0.5, 0.5, 0.2]
    bone_count: [0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 2.0, 3.0, 4.0, 5.0]
```

The parameters in the beginning of the training start at 0.0, 3.0, 0.0; and change to 0.0, 2.5, 0.0 respectively when the Agents reach the reward of -0.1, and so on. The min_lesson_length makes sure they take part in at least 100 lessons before changing to next lesson. As a result, it would prevent the difficulty increase immediately, if the agent somehow gets lucky with a high score.

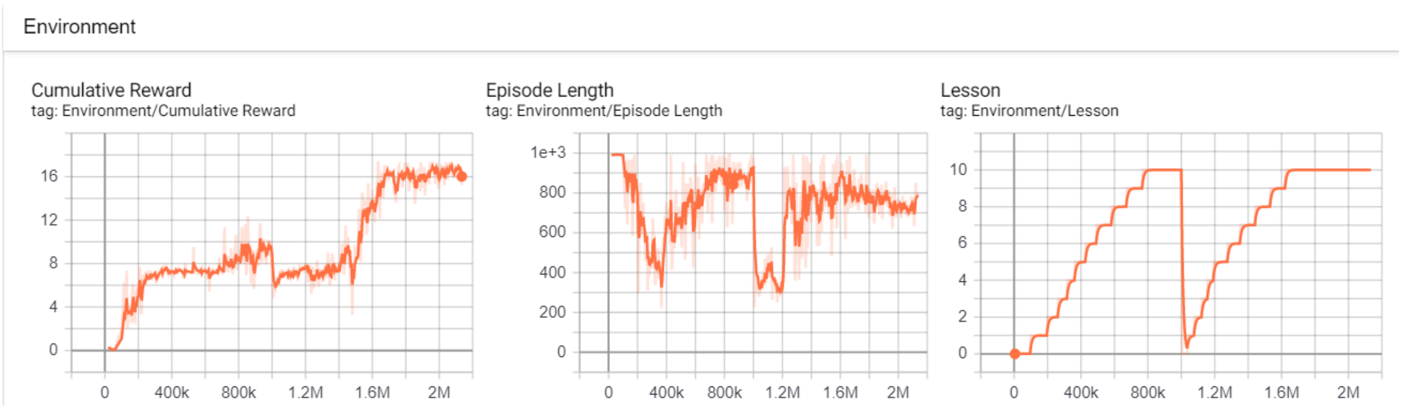
4. Inference

After finishing training, you can set up the penguins to perform inference, which means they will make decisions using the neural network that is previously trained. You just need to import the .nn file to Unity, then drag it to the Model part under Behavior Parameters.


Model PenguinLearning (NNModel)

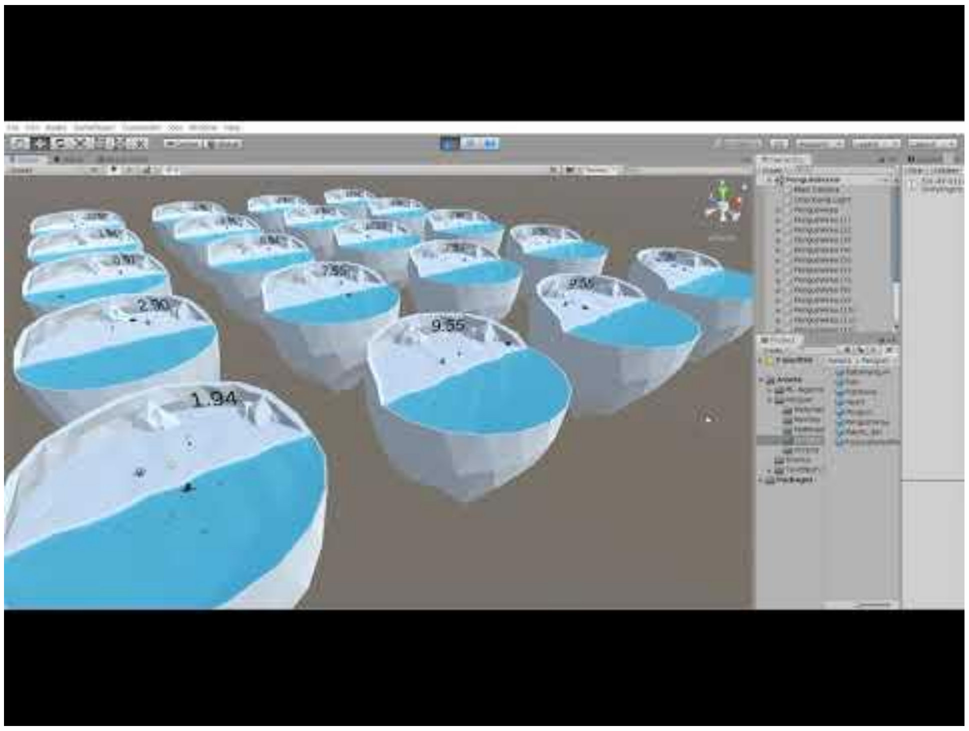
III. End Result

Unfortunately, there is no screenshot in the console while training. However, thanks to TensorBoard, the statistics are saved during learning session and shown by charts.



The horizontal axes are showing the number of steps. There are actually 2 training sessions, linked at 1M steps. The second training accidentally restart the curriculum, so the Lesson chart has a significant drop down in the middle. The other 2 also has a drop, but quickly return to previous state.

The Cumulative Reward is likely the most important one to check the Agent's intelligence. There are 2 big milestones, at around 100k steps and 1.5M steps. Those show that the Agents have big success in learning behavior. After 2M steps, there was no much possibility for improvement, so I stopped the training with our desired result.



IV. Reference

- ML-Agents Python Setup with Anaconda
- Reinforcement Learning Penguins | Unity ML-Agents
- Unity Machine Learning Documents