Readme for Robot Arm ML Agent Unity
CTI One Corporation
Santa Clara, CA

2021-3-14	Created this document	Chee Vang
2021-3-26	Tested and verified with update	HL and Yusuke Yakuwa
2021-4-9	Updated with more details	Chee Vang
2021-5-7	Reorganized the document with sections; Added Section I, and Section VI and VII; added Table of Contents	HL
2021-5-14	Update and Added TensorBoard instructions	Chee Vang

https://github.com/rkandas/RobotArmMLAgentUnity
https://medium.com/xrpractices/how-to-train-your-robot-arm-fbf5dcd807e1

TABLE OF CONTENT

I. INTRODUCTION

This readme covers the following 4 parts for Unity based AI training and simulation.

The first part is about loading and/or creating a project, and select the right ML library function, for example from the following item 1 to 5;

The second part is about define the right visualization model of the robot, which is basically the graphics part, from item 6 to item 9 in the following description;

The third part is about training from item 10 to item 13 with ML library and the graphics model as described in the following sections.

Then in the last part, e.g., the 4th part, we describe playing back of the training result in Section V, analysis and interpretation of the trained result in Section VI, coding and customer algorithm in Section VII.

II. DOWN LOAD ML PACKAGE AND PROJECT CONFIGURATION

This project requires Python 3.7 environment.

1) Create a conda environment with Python 3.7

```
conda create --name unityenv python=3.7
conda activate unityenv
```

2) Clone the github

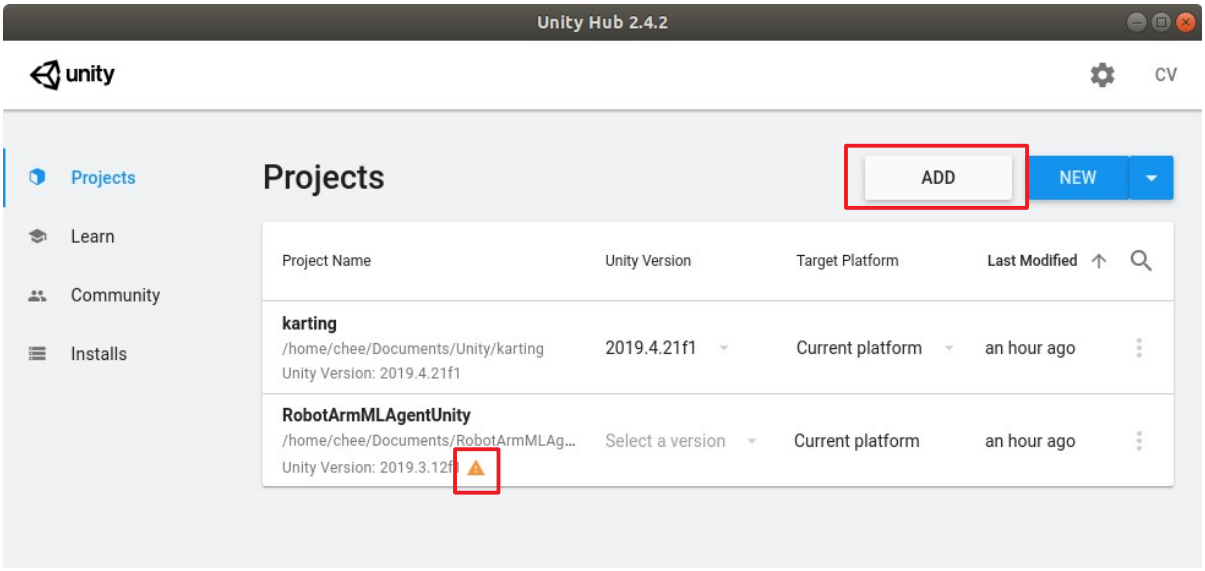
```
cd ~/<where you want to save the project>/
git clone https://github.com/rkandas/RobotArmMLAgentUnity.git
```

3) Install ML-Agents (Use pip3 instead if pip does not work)

```
pip install mlagents
```

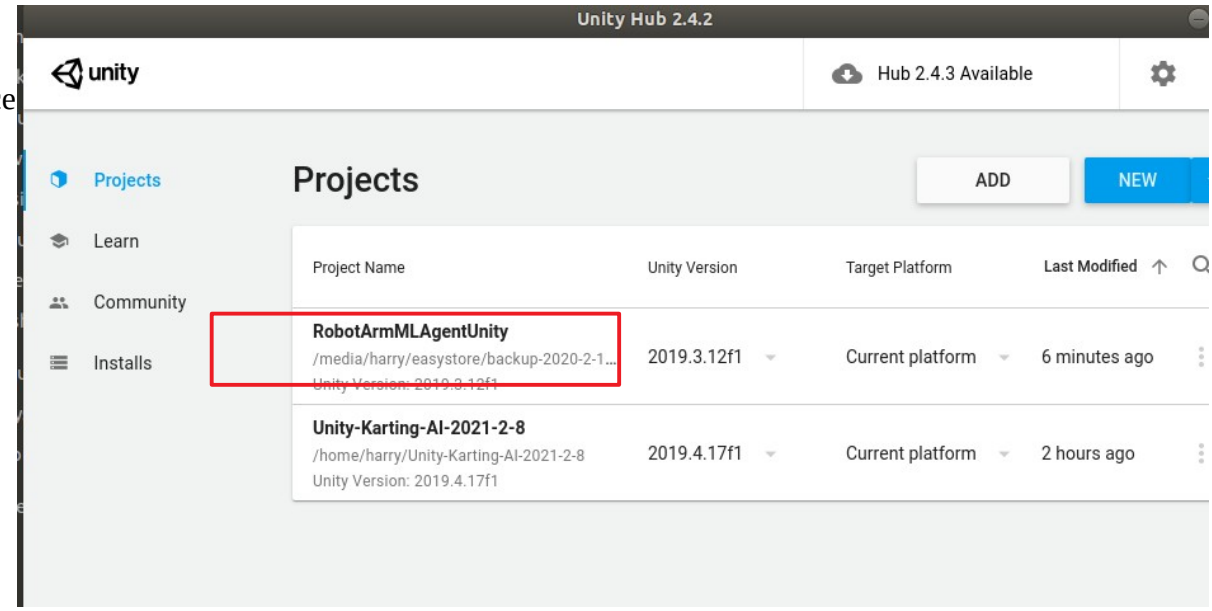
4) Open Unity Hub. **Add** a new project and select the folder **RobotArmMLAgentUnity** that was just downloaded.

5) There might be a warning symbol on the project about the Unity version. You can ignore the warning and select a version, or install Unity 2019.3.12f1.

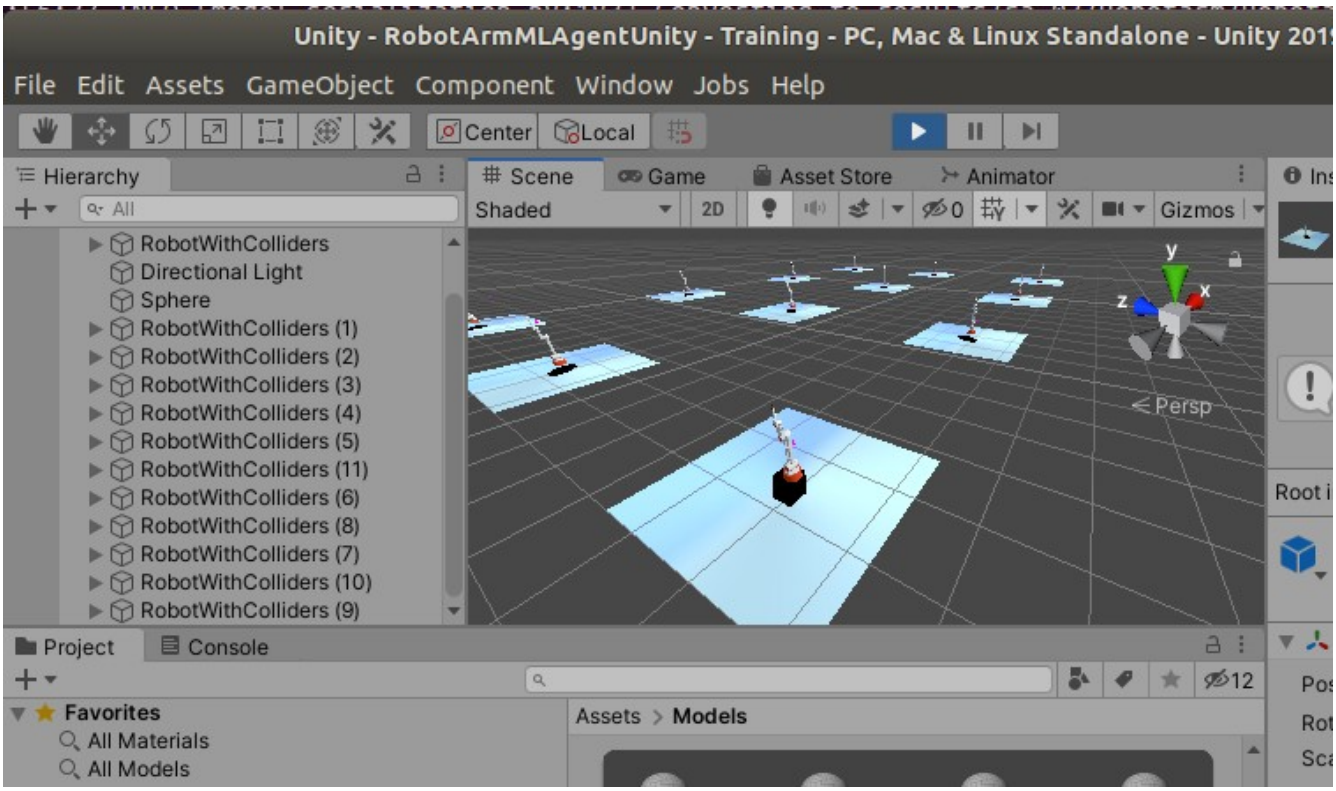


Note: To start the project, from the gui of UNITY, double click on the project name that you have created, as illustrated below, unity will then open that project for you. This may take a few minutes for the system to load.

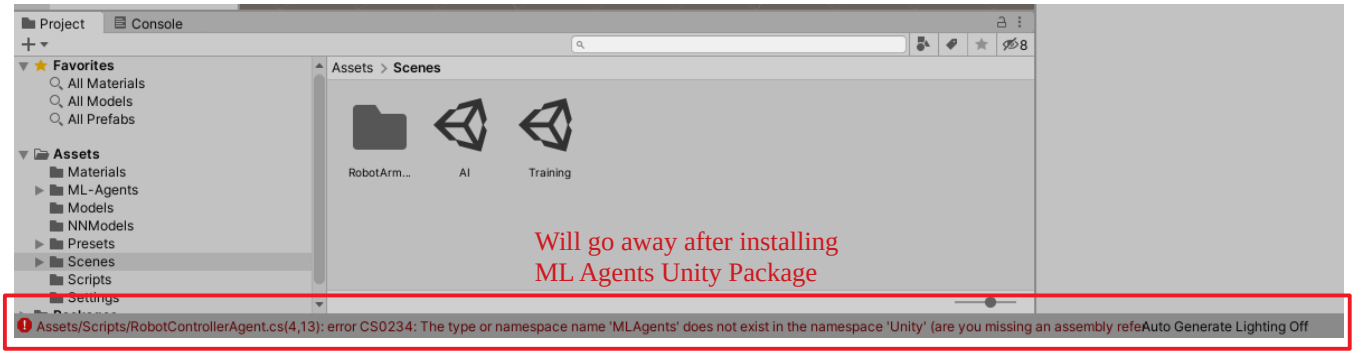
Once the

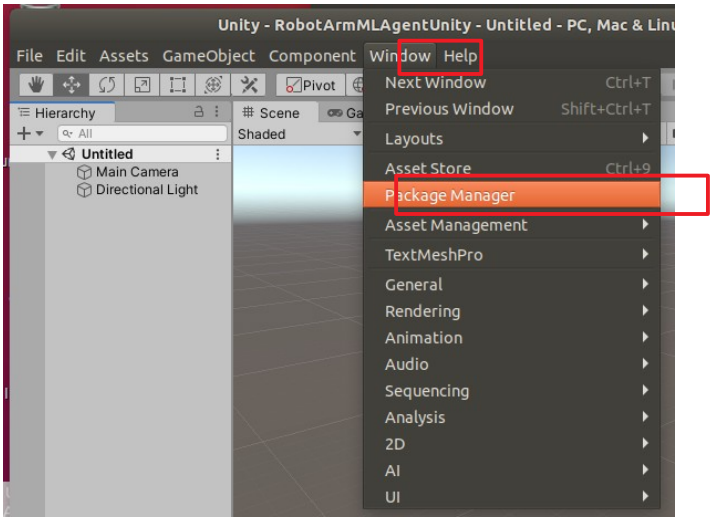


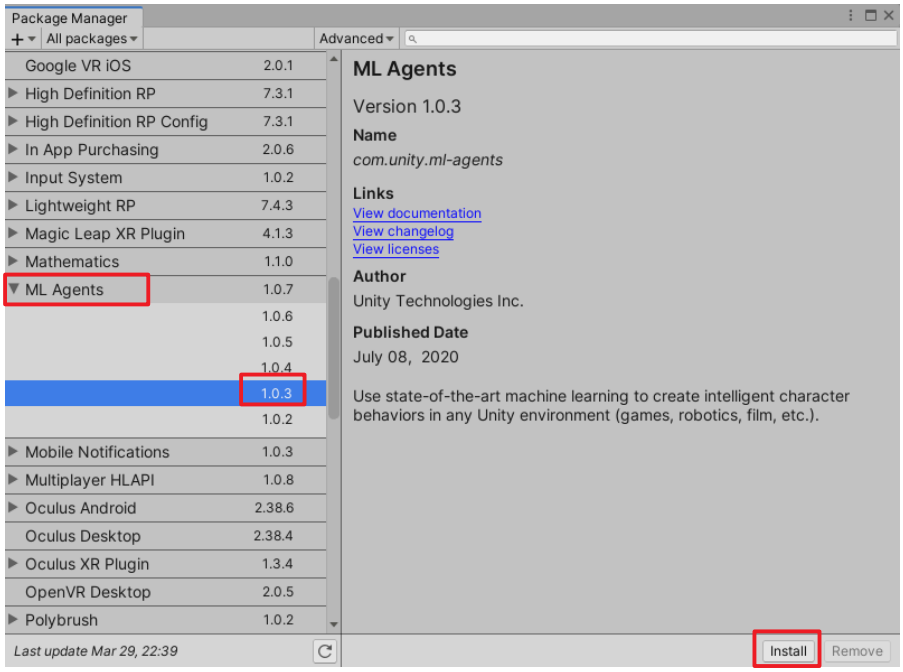
project is opened, you will see the gui screen as shown below, you may not have the robots in your gui, I have it here because I have already configured it.



6) When the project is first opened, there will be a few errors. Just import ML Agents by going to **Window** >> P**ackage Manager** and on **ML Agents** click on **See all version** to import/install version **1.0.3**.

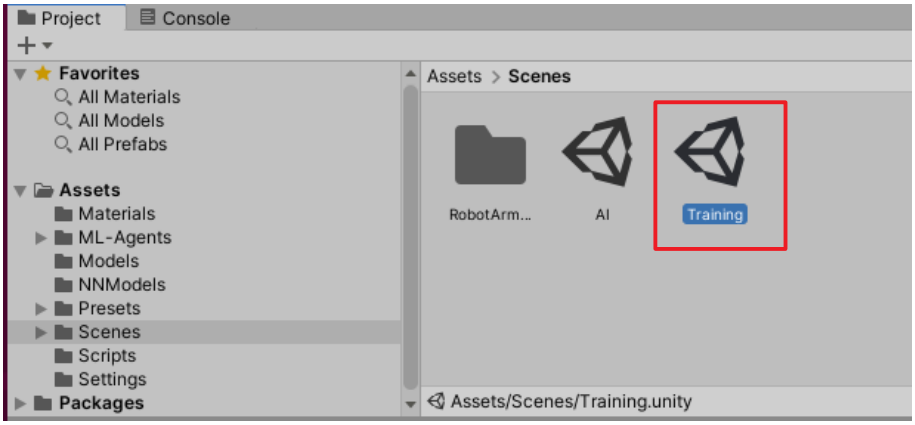




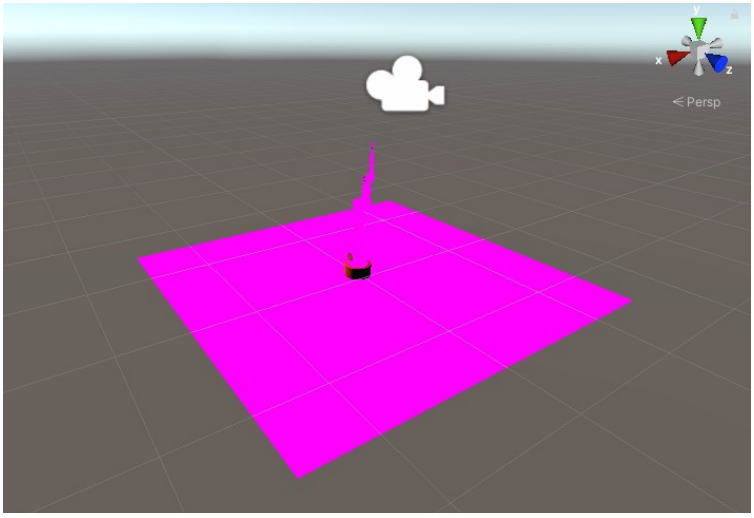


III. CREATE VISUALIZATION MODEL OF THE ROBOT

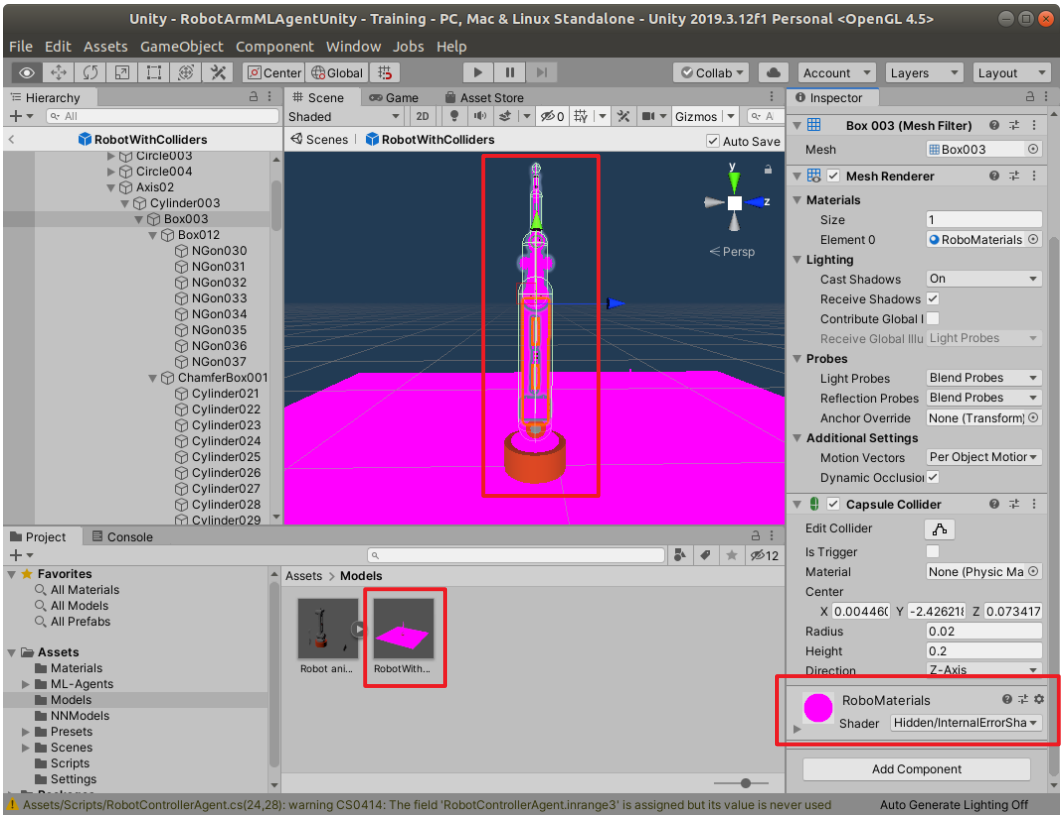
7) In the **Project** window, go to **Assets** >> **Scenes** and open the **Training** scene with 12 robot arms.



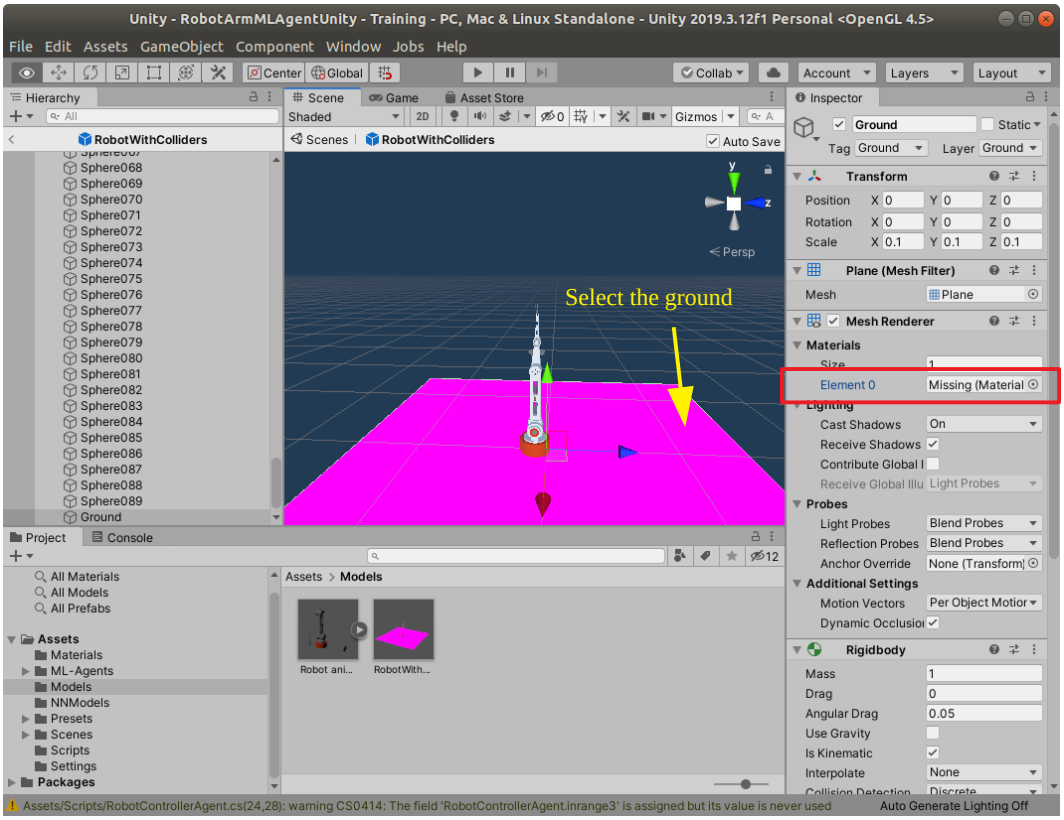
8) An issue will appear where the prefabs are pink like the figure below. If not, go to the step (9). Otherwise, continue.



8.1 In the **Project** window, go to **Assets** >> **Models**.
8.2 Double click on **RobotWithColliders.prefab** in the **Project** Window (shown below)
8.3 Click on the robot arm, it will be highlighted. On the right side, is the Inspector panel.
8.4 In **Inspector** >> **Robot Material**, change **Shader** to **Standard**.
8.5 Do the same for other pink items: bottle and grabber.

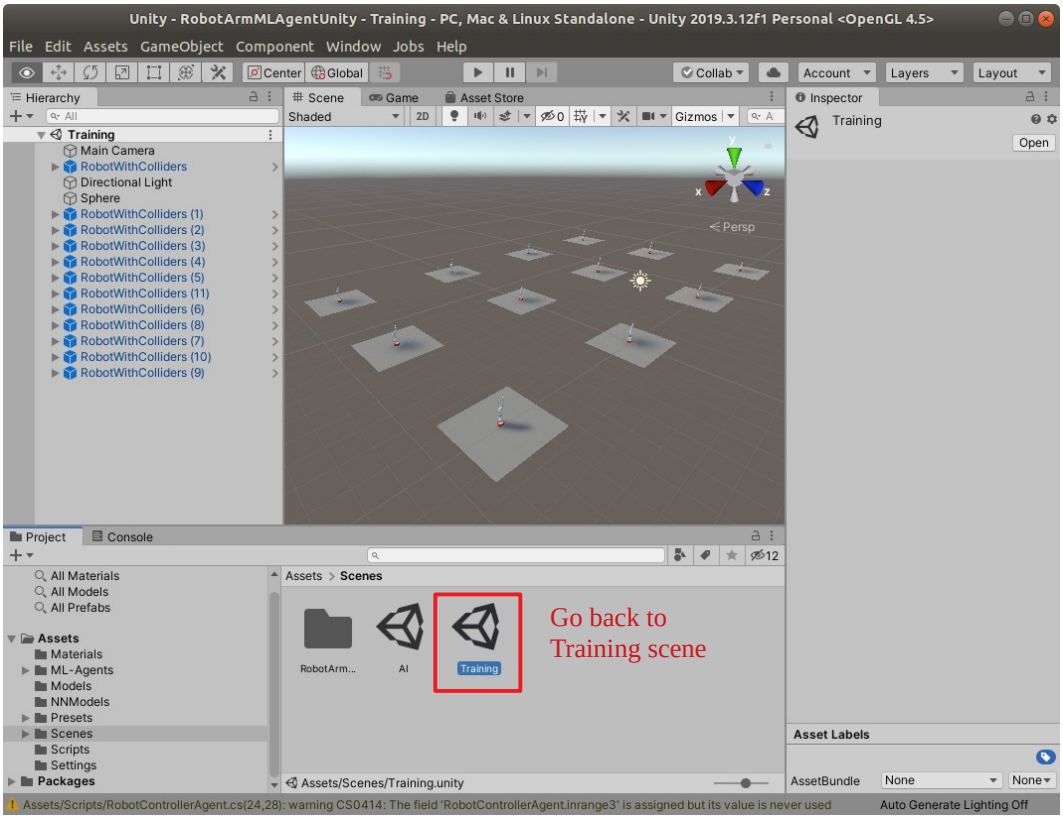


8.6 To change the ground: select the ground and change the material in **Inspector** >> **Mesh Renderer** >> **Materials** >> **Element 0** to another material

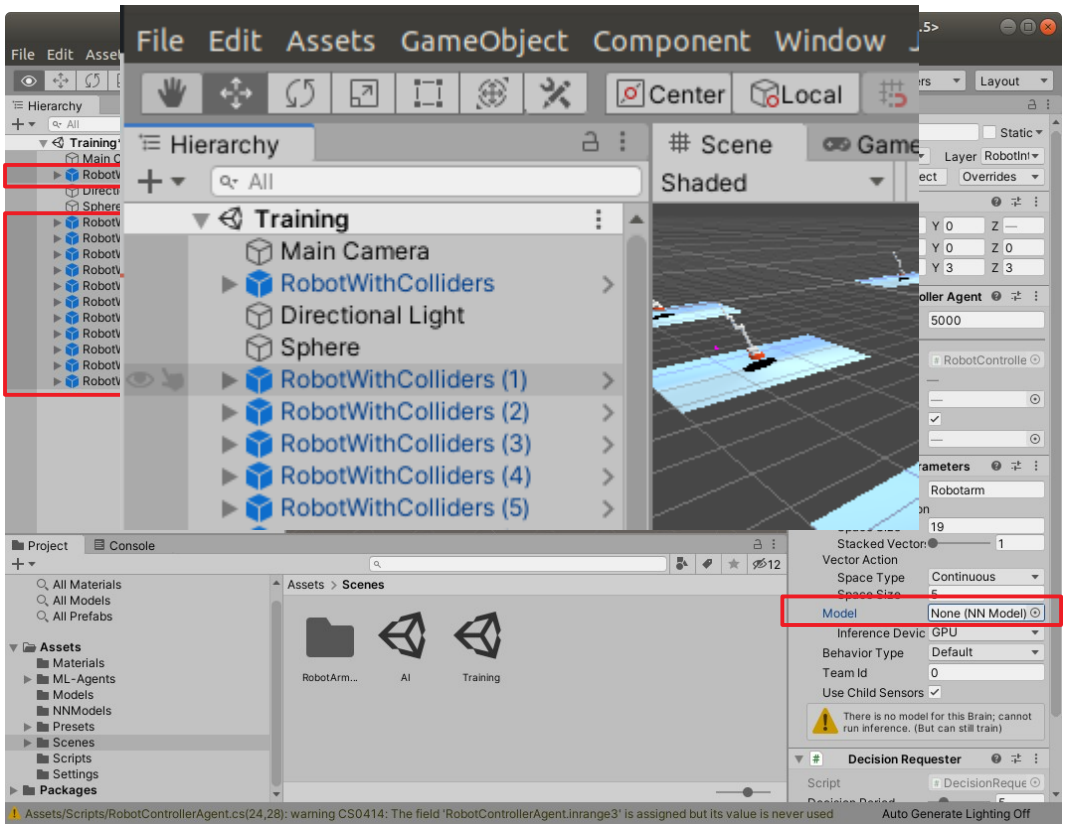


After changing the materials/shader, this should also affect the prefabs in the scenes we will use. Go back to **Project** >> **Assets** >> **Scenes** >> **Training** and the robot arm should be the selected color.

9) Select all the



RobotWithColliders in the **Hierarchy** and change the model to **None** in **Inspector** >> **Behavior** >> **Model**. (Hold the [Shift] key to select more than one item in the Hierarchy. Use [Ctrl] key to select/deselect one item at a time.)



IV. TRAINING

The training will have to be operated both inside Unity GUI and outside Unity GUI, at your downloaded ML folder. Once you enter a training command at the terminal opened at your downlonad ML package folder, you must go back to Unity GUI to click “play” to start the training correctly. So, the explanation is given below.

10) The **trainer_config.yaml** file (found in the project folder RobotArmMLAgentUnity) can be modified to change some of the configurations for **Robotarm**:

Note: Changing any of these parameters will change the result, accuracy and model

- **batch_size :**
- **hidden_units** : number of nodes per layer in NN
- **num_layers** : number of layers
- **max_steps** : number of training steps (Mine was changed to 1.0e6)

11) In order to start the training, now you will have to go outside the Unity GUI. So go to the folder where you ML package is downloaded from the github, for example in my case shown below:

/RobotArmMLAgentUnity$ tree -L 1

```
.
├── Assets
├── Library
├── LICENSE
├── Logs
├── Packages
├── ProjectSettings
├── README.md
├── results
├── Temp
├── trainer_config.yaml
├── trainer_sac_config_for_1_-and_2.yaml
└── trainer_sac_config_piecewise.yaml
```

7 directories, 5 files

11a) Note that I have 3 yaml configuration files for training. You would have just one yaml file, because your downloaded git package only have trainer_config.yaml. Now open the terminal at this folder, you must enter the online command “mlagents-learn” to start the training process as follows

```
$mlagents-learn ./trainer_config.yaml --run-id ra_01
```

Note:

- option can be added after `--`, so add rund-id to defines the trained result in folder ra_01, you can use any meaningful folder name, such as my-first-test.
- option `--force` to overwrite training model with name specified by run-id

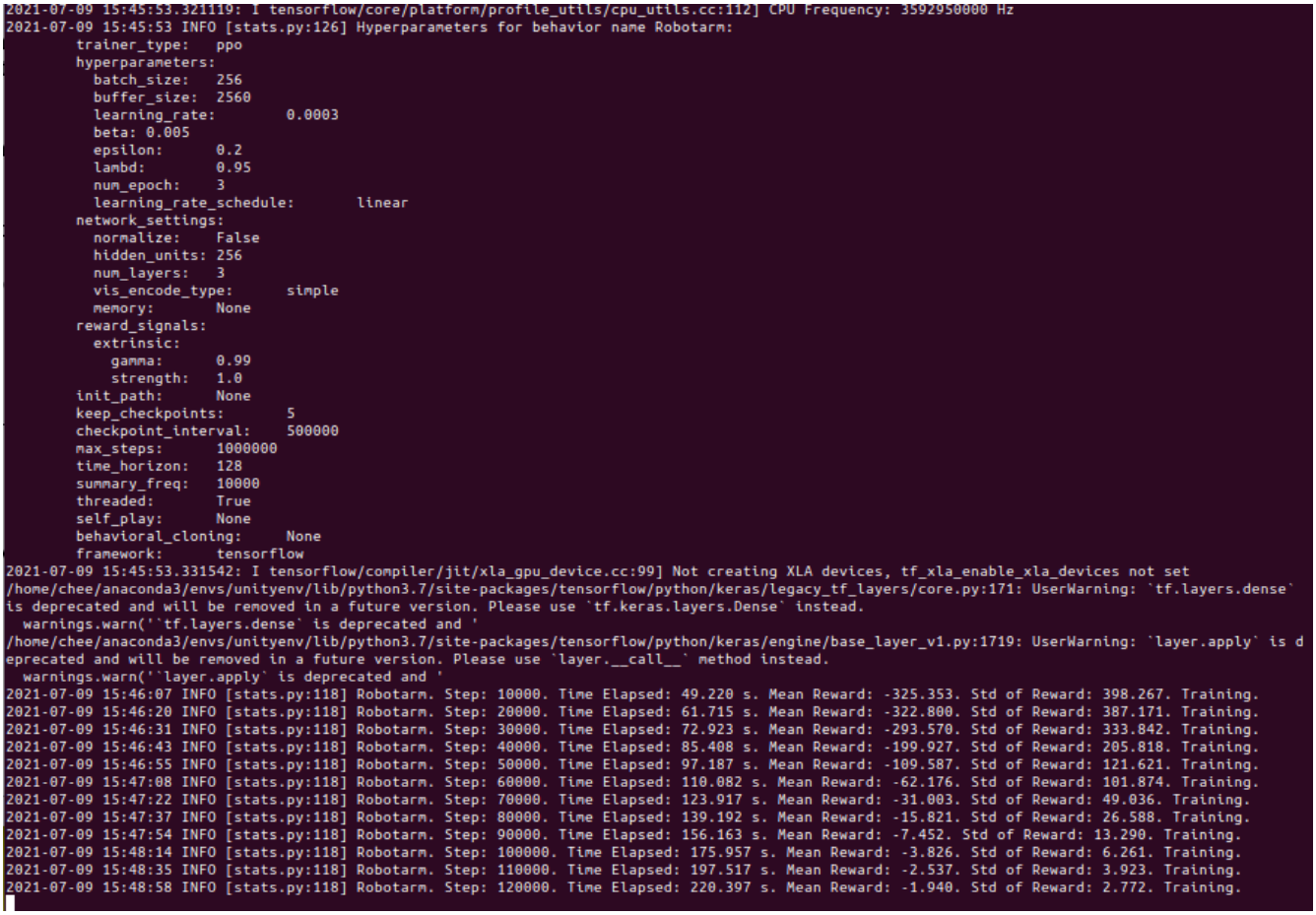
```
$mlagents-learn ./trainer_config.yaml --run-id ra_01 --force
```

- `option--resume` to continue training from previously paused/stopped model with name specified by run-id

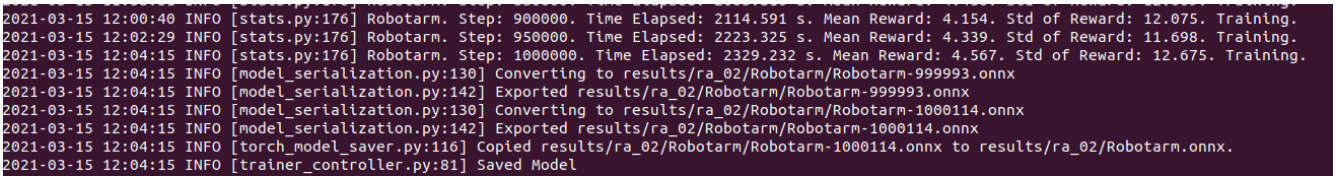
```
$mlagents-learn ./trainer_config.yaml --run-id ra_01 --resume
```

11b) Go back to Unity and press the **Play** button to start the training. It will take several minutes.

11c) Details of the training configurations will be displayed onto the terminal and the training statistics during the training as shown below.



12) Once the training is completed, the terminal should have the following:



Note (HL 2021-3-27) checking the printed message in the console, you will see
(1) this following line of message shows the result converted to the results folder and sub-folders in a file with file extension ".onnx";

2021-03-26 18:17:07 INFO [model_serialization.py:183] Converting to results/ra_01/Robotarm/Robotarm-1060.onnx

(2) the 2nd line of message shows the result is exported as follows:

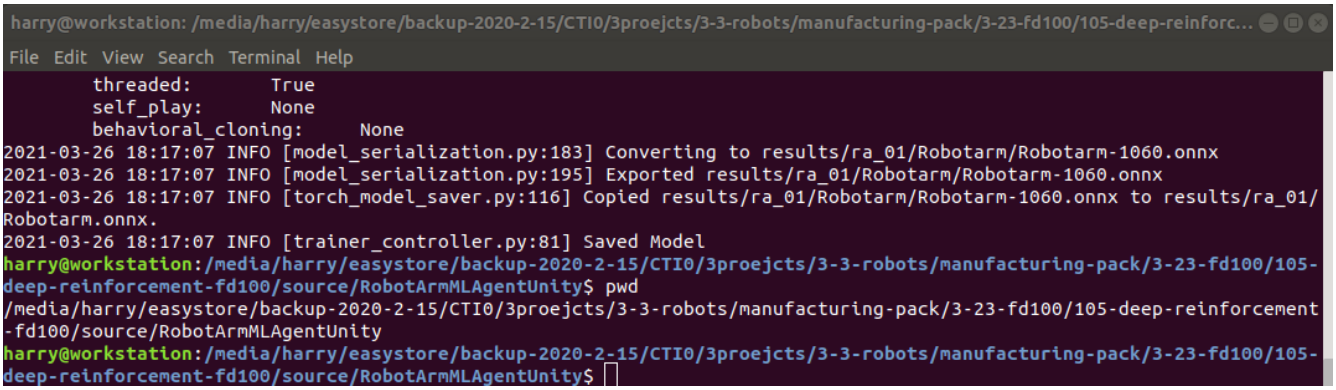
2021-03-26 18:17:07 INFO [model_serialization.py:195] Exported results/ra_01/Robotarm/Robotarm-1060.onnx

(3) the 3rd line of the message shows the copied result.

2021-03-26 18:17:07 INFO [torch_model_saver.py:116] Copied results/ra_01/Robotarm/Robotarm-1060.onnx to results/ra_01/Robotarm.onnx.

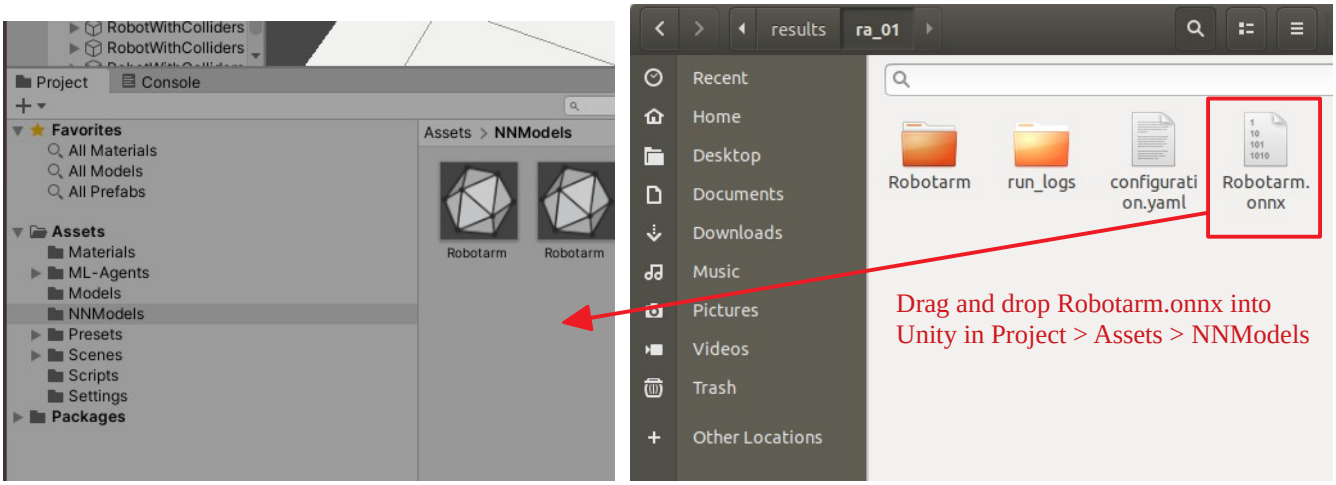
(4) finally, the message for the saved model.

2021-03-26 18:17:07 INFO [trainer_controller.py:81] Saved Model

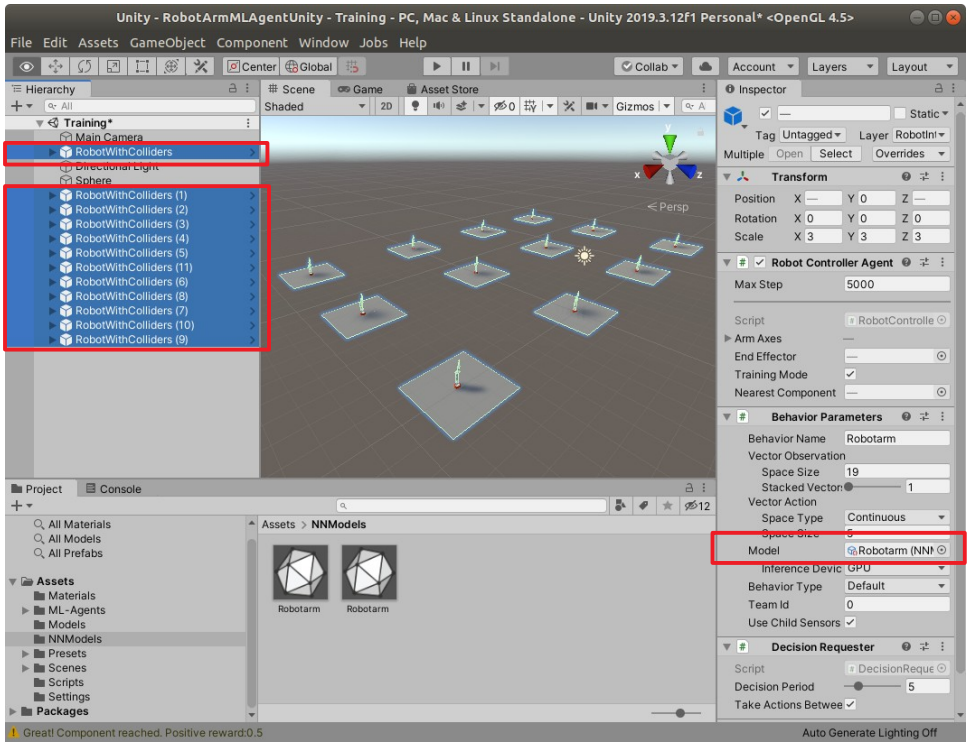


V. PLAY BACK ANIMATION OF THE TRAINED RESULT

13) To play the trained model for visualization purposes, drag and drop the trained model in **RobotArmMLAgentUnity/results/ra_01/Robotarm.onnx** to Unity in **Project** > **Assets** > **NNModels**



14) Select all the **RobotWithColliders**. Drag and drop the new **Robotarm.onnx** into **Model** in **Inspector** >> **Behavior Parameters** >> **Model**



Note: if you just select the one you want, then the Unity will only play with that selected robot

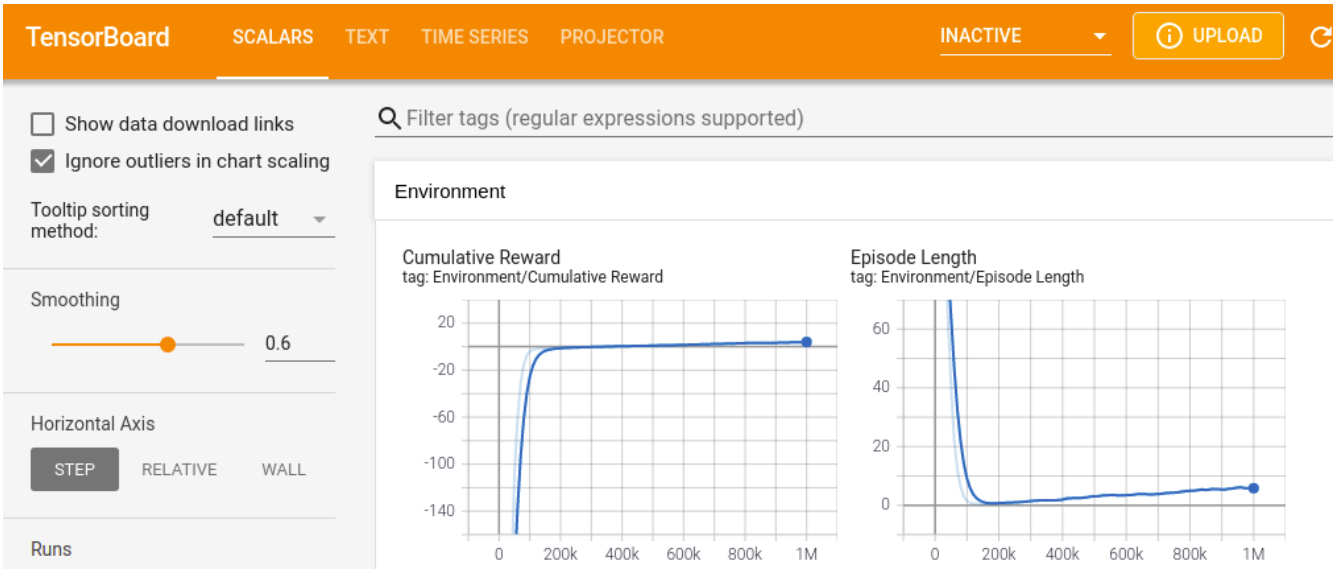
15) Press **Play** button to watch the animation of the trained result.

16) Training results can be displayed onto a graph through TensorBoard by entering into the terminal

```
tensorboard --logdir results
```

where `--logdir` specifies the directory of the trained results, in this cases they are located relative to the project in the `results` folder.

A link will be provided where the graphs can be displayed onto a web browser. Below are graphs of the training with max_steps = 1.0e6.



VI. DATA ANALYSIS OF THE TRAINING RESULT

Once the training is completed, a new folder is created and it holds the following files and 2 folders:

```
/results/sac-1-2$ tree -L 1
.
├── configuration.yaml
├── Robotarm
├── Robotarm.onnx
└── run_logs

2 directories, 2 files
```

Note:
1. The file configuration.yaml gives you the detailed configuration history of the training model;
2. The file Robotarm.onnx gives you the animation file needed to run Unity visualization;
3. The folder Robotarm gives your 4 files:

```
/results/sac-1-2/Robotarm$ tree -L 1
.
├── checkpoint.pt
├── events.out.tfevents.1620456497.workstation.12612.0
├── Robotarm-7179.onnx
└── Robotarm-7179.pt

0 directories, 4 files
```

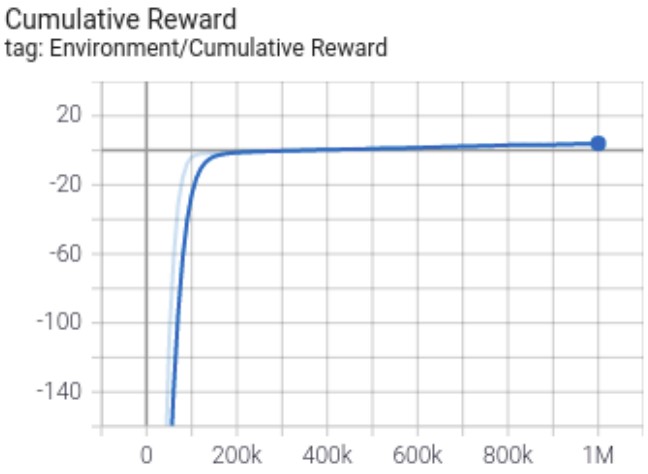
4. The folder run_logs gives you 2 files:

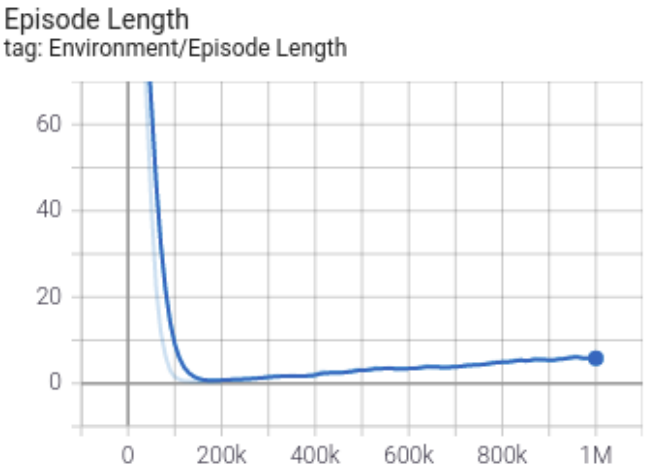
```
/results/sac-1-2/run_logs$ tree -L 1
.
```

```
├── timers.json
└── training_status.json
```

0 directories, 2 files

From TensorBoard:





Details of each graphs is given in the table below or found at https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Using-Tensorboard.md. The cumulative rewards starts at a negative value but increases to a positive value. This values continues to increase but the rate at which it is increasing is stablizing. This corresponds to a successful training as explained from ml-agents documentation. For the Episode Length, the length of each episode is longer at the beginning because the robot arm has not been trained enough. After 150k steps, the episode length is much smaller in comparison to before 150k steps since it requires less steps to reach the target.

Graph	Description
Cumulative Reward	mean cumulative reward (increases if training successful)
Episode Length	mean length of each episode
Is Training	a booleon to indicate if agent is updating model
Policy Loss	mean magnitude of policy loss function which correlates how much policy is learning (decrease if training successful)
Value Loss	mean loss of value function update which correlates how well model is able to predict value of each state (increases while agent learning and decreases once reward stablizes)
Entropy	describes randomness of model (slowly decreases if training is successful, and if it decreases too quickly, increase *beta* hyperparameter)
Entropy Coeff	determins relative importance of entropy term
Extrinsic Reward	corresponds to mean cumulative reward for each episode
Extrinsic Value Estimate	mean value estimate for all states visited by agent (increases if training successful)

Learning Rate	describes how large a step the training algorithm takes as it searches for optimal policy (should decrease over time)

VII. CODING AND CUSTOMER ALGORITHM

Add step by step coding and customer algorithm here.

(END)

July 9, 2021
assets.unity.com
https://assetstore.unity.com/summer-sale?gclid=CjwKCAjw55-HBhAHEiwARMCsztmk-UTdmu9z26nF4rnXZ6h6KXWv_StHbsfqaHabKwE0_4M7yAzfUxoC3_oQAvD_BwE