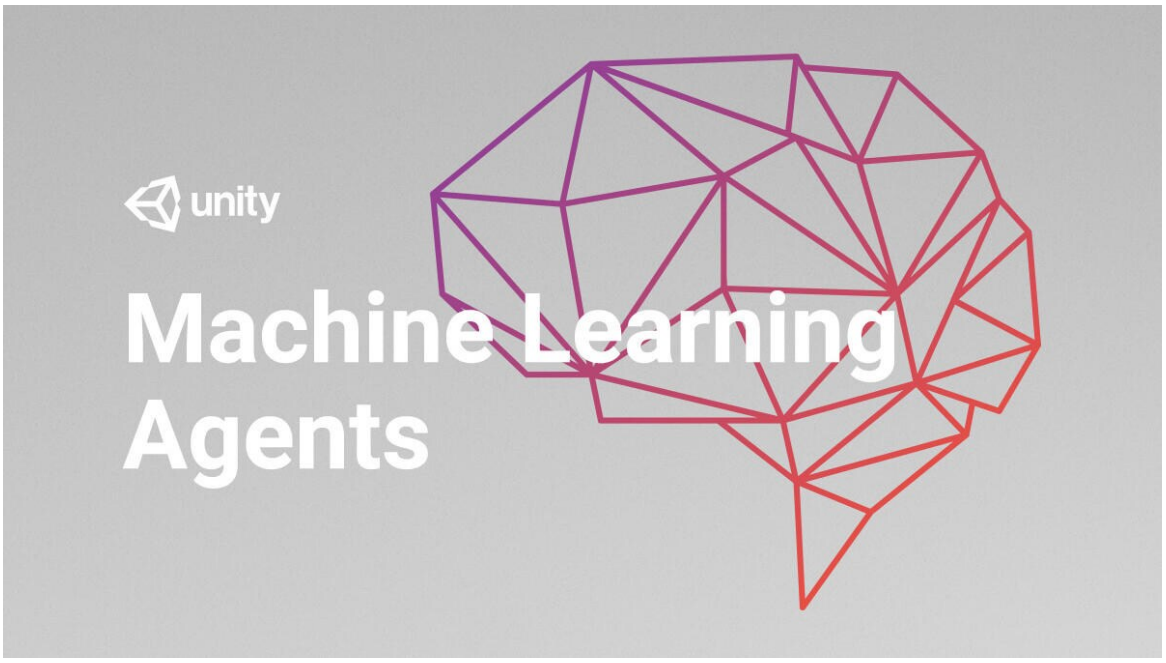


Unity ML Agents

The Unity Toolkit used to train agents

By Lea Wölfl, Xabier Arriaga & Jon Ander Ruiz
Advanced Techniques in Artificial Intelligence

Donostia, 09-22-2021

Contents

Introduction

Unity ML-Agents toolkit is a new plugin based on the game engine Unity that allows us to use the Unity Game Engine as an environment builder to train agents. But first we have to define what Unity is.

"Unity is a cross-platform game engine with a built-in IDE developed by Unity Technologies. It is used to develop video games for web plugins, desktop platforms, consoles and mobile devices".

IDE stands for "Integrated Development Environment", which means that in this case unity is a union of a game engine, that allows game created to run (hence to be played) in different environments, an application where the "visible pieces" of a game can be put together (the IDE) with a graphical preview and using a controlled "play it" function, and a code editor.

Now that we have defined the Unity engine which is a useful tool for video games development, not only because of the environment itself but also because of all its possible tools. We will move forward to explain how this toolkit in particular works, and we will show a demonstration about it.

Before starting, it needs to be said that to understand the use of this toolkit it is needed to have some knowledge in the subject of reinforcement learning, and also on the proper basic use of the Unity software.

How does it work?

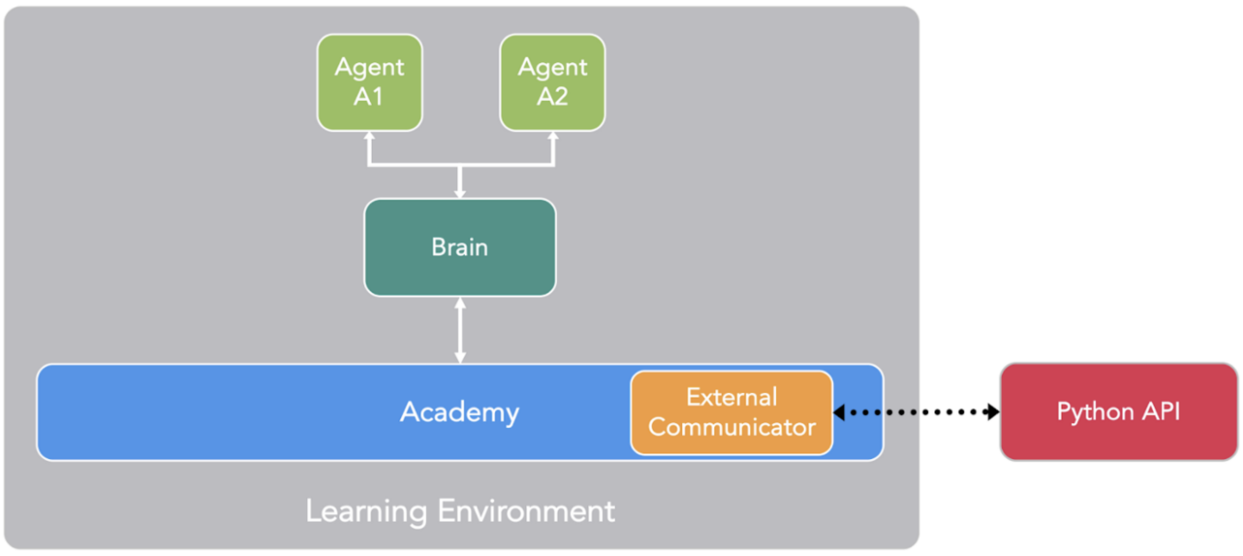
The Unity Machine Learning Agent Toolkit is based on Python and TensorFlow.

Unity ML-Agents can be trained using multiple techniques, for example imitation learning, neuroevolution or others. The method used here however, is Deep Reinforcement Learning. The method of Reinforcement Learning can be compared to training a dog. The agent can choose from different actions and by executing one or several actions it can achieve a specified task. If the task was executed successfully the agent gets a reward. Therefore it learns to try and do the specified task over and over again.

The components

The Unity ML-Agents itself consists of three main components. The Learning Environment, the external communicator and the Python API.

The Learning Environment contains the different visible components, meaning the unity scene and the elements placed inside the scene.
With the Python API we are able to implement different algorithms in order to train and test the agents. The communication between the API and the Learning Environment is done by the External Communicator.



Source: Unity ML-Agents Documentation

Inside the Learning Environment we then have different components. The Agent is the actor that in the end will execute the actions that we are training. The Agent itself is not optimizing its behavior. That part takes place inside the Brain.
The Academy is the element that manages the agents and the decision-making. The Academy receives the requests from the Python API and then controls the agents via the brain. Similarly to how a puppet would be controlled by the puppet master.

The Reinforcement Training Loop

The Reinforcement Training Loop is executed by the academy. At first the agent starts at the initial position and collects all the information about his environment.

Then it executes an action, for example it takes a step. By doing so the environment is changed into a different state. The agent then evaluates the new state of the environment and then either collects a reward if the goal was reached or it takes another action.

This loop is repeated until the agent either:

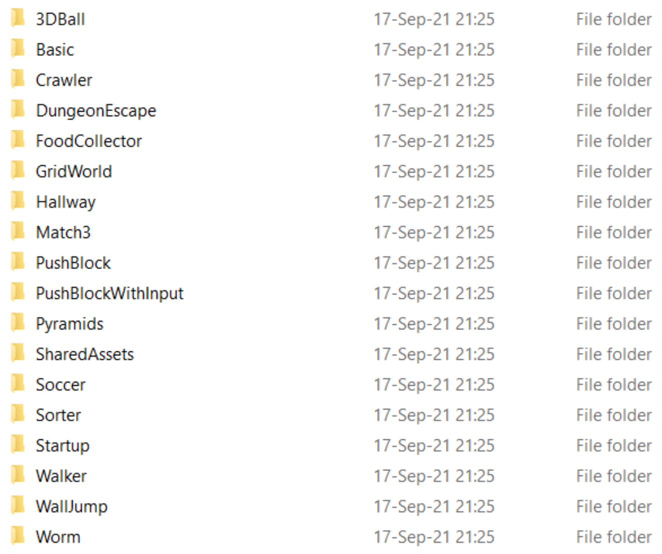
- gets the reward
- or the number of maximum steps is reached.

Afterwards the environment is reseted and the loop starts again.

The goal is of course to get the maximum amount of rewards and therefore the agents learn to execute the specified task.

Demonstration

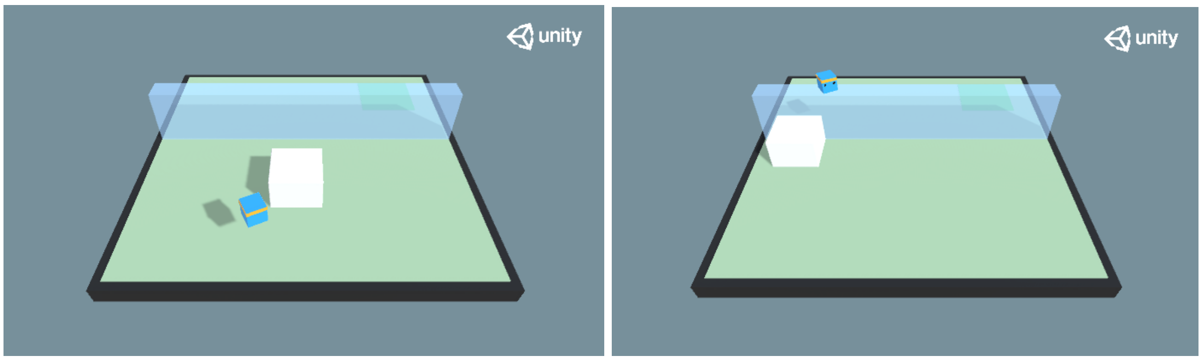
For this demonstration, the examples on the Unity ML Agents github page were used, as we could use them immediately after downloading them. You can obtain them by downloading their github repository here.



3DBall	17-Sep-21 21:25	File folder
Basic	17-Sep-21 21:25	File folder
Crawler	17-Sep-21 21:25	File folder
DungeonEscape	17-Sep-21 21:25	File folder
FoodCollector	17-Sep-21 21:25	File folder
GridWorld	17-Sep-21 21:25	File folder
Hallway	17-Sep-21 21:25	File folder
Match3	17-Sep-21 21:25	File folder
PushBlock	17-Sep-21 21:25	File folder
PushBlockWithInput	17-Sep-21 21:25	File folder
Pyramids	17-Sep-21 21:25	File folder
SharedAssets	17-Sep-21 21:25	File folder
Soccer	17-Sep-21 21:25	File folder
Sorter	17-Sep-21 21:25	File folder
Startup	17-Sep-21 21:25	File folder
Walker	17-Sep-21 21:25	File folder
WallJump	17-Sep-21 21:25	File folder
Worm	17-Sep-21 21:25	File folder

In the previous picture we can see the examples included in the repository.
We tried “Walker” and “WallJump” but we are going to explain in depth only “WallJump”.

WallJump



In this case, the agent (blue) must reach the green zone on the other side of the wall. Sometimes the agent can jump across the wall by itself, but when this is not possible, the agent must use the white box on its side of the wall. The agent uses this box by moving it to the wall and then jumping on it so it can jump across the wall.

The agent will learn two different policies, depending on the height of the wall:

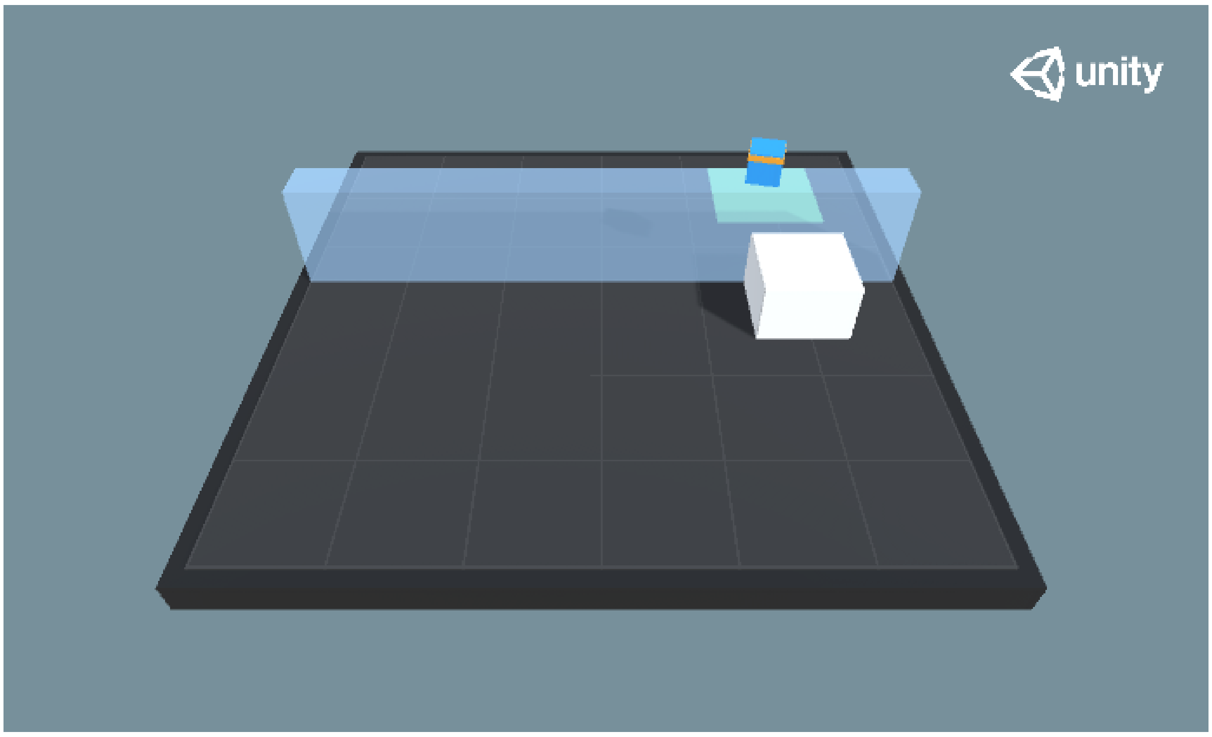
- SmallWallJump: No wall and low wall cases.
- BigWallJump: High wall situations.

The reward system is as it follows:

1.- -0.005 for every step so it moves as fast as possible to the green zone
2.- +1 if the agent reaches the goal
3.- -0.01 if the agent falls of the platform

In terms of observation, it uses 14 raycasts that each detect 4 possible objects. The global position of the agent is also used.
The action space has 4 branches:

- Forward Motion: UP / DOWN / NO ACTION
- Rotation: ROTATE LEFT / ROTATE RIGHT / NO ACTION
- Side Motion: LEFT / RIGHT / NO ACTION
- Jump: JUMP / NO ACTION



How to prepare our system to use this toolkit

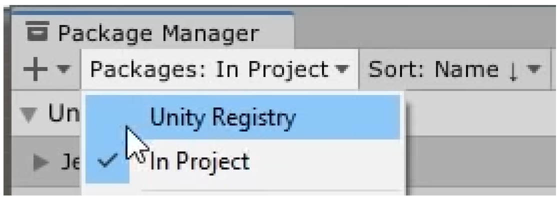
In order to train your own AI using ML Agent Toolkit, first you should install the Python and Pytorch versions that are in the documentation in GitHub, as the project is constantly being updated. They recommend using Python 3.6 and 3.7.
Then, it's recommended to create a virtual environment using python, so we can have our project separate from the others and use the necessary python packages without causing any conflicts with other projects.
For this we could use in Windows:

```
python -m venv venv
```

Then, we should install PIP.

Next, we install ml-agents. In case this gives some compatibility errors, we can fix them by installing the asked dependencies. This usually happens with the GPU dependencies.

Finally, in Unity we must install the ML Agents package. For this we open the package manager (window → Package Manager) and be sure to go into the Unity registry, as shown in the picture below. Then find the ML Agents package and choose either the stable version or the newest preview one.



With this we are ready to start our own project.

Bibliography

- ➢ [GitHub repository]: https://github.com/Unity-Technologies/ml-agents
- ➢ [Article]: An Introduction to Unity ML-Agents | by Thomas Simonini
- ➢ [Video]: ▶ How to use Machine Learning AI in Unity! (ML-Agents)
- ➢ [Article]:Why is Unity so popular for videogame development?
- ➢ [Video]: Reinforcement Learning: Crash Course AI
- ➢ [Video]: Unity at GDC