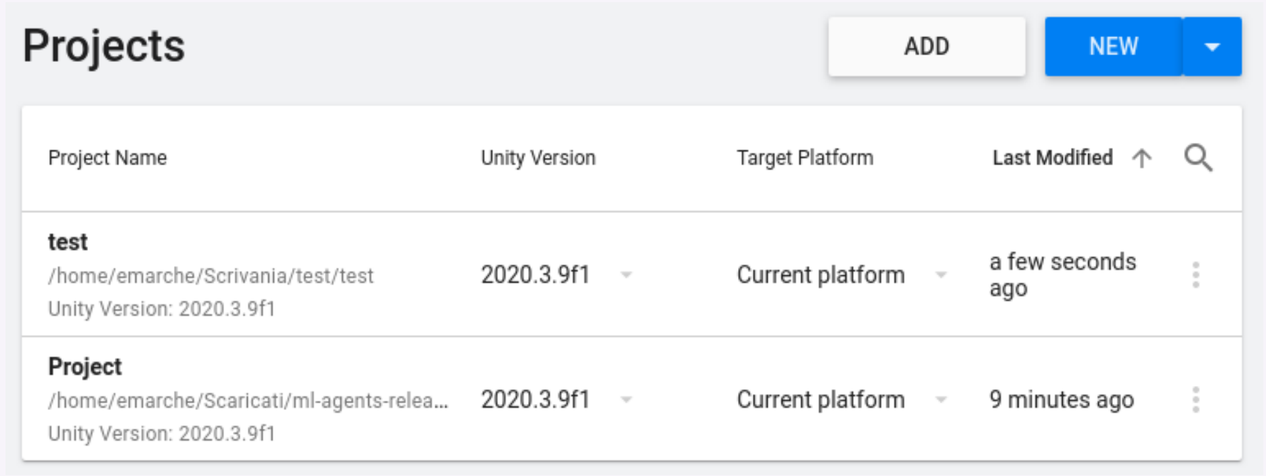
[Lab 5]

Unity and ML-Agents Toolkit

Getting Started

- From the Hub, select "Add":
  - Navigate to the cloned repository
  - Add the "Project" Folder.

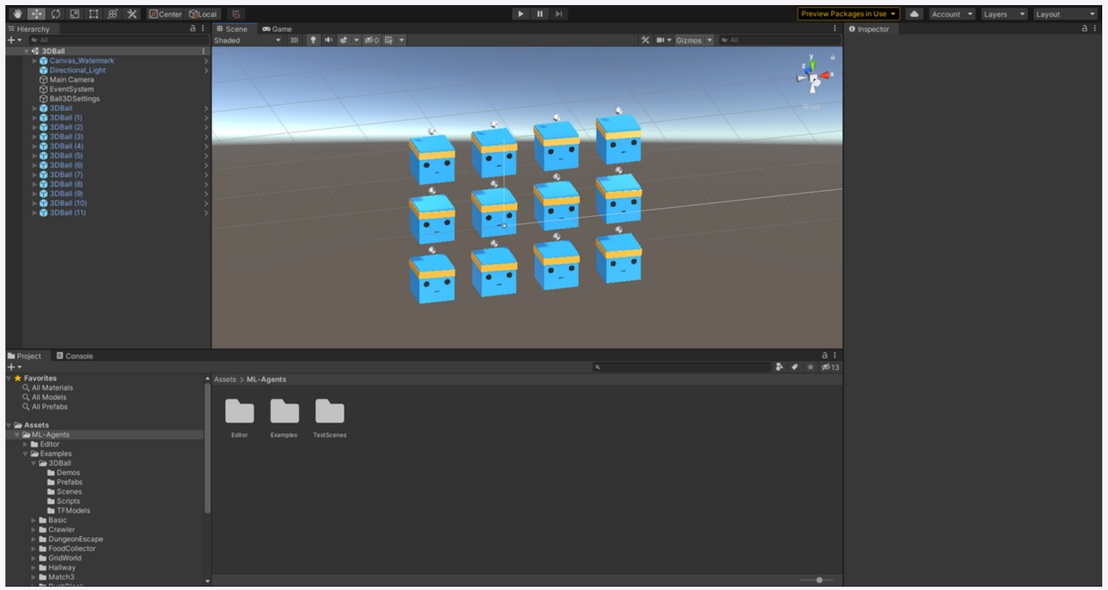


  - Select the installed Unity Version and open the project.

Unity

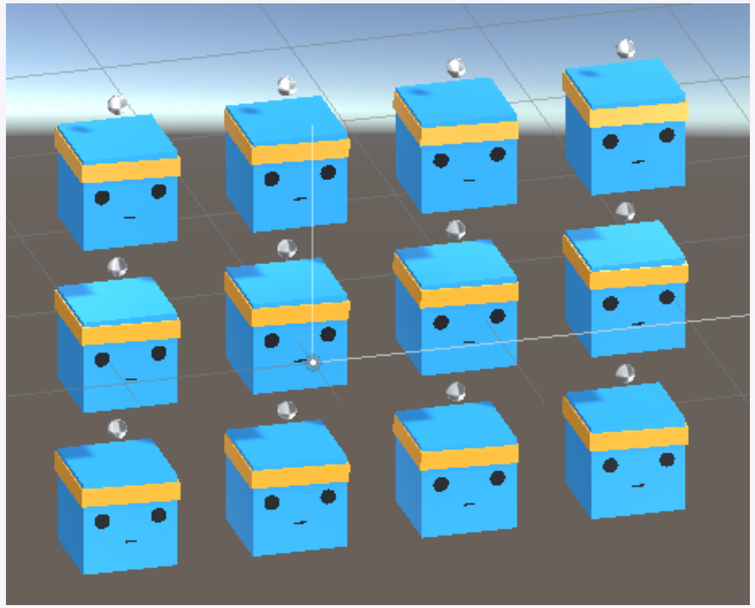
There are several windows in the Unity editor:

- The Project window displays the library of Assets that are available to use in the Project (i.e., we should have ML-Agents here).
- The Scene/Game windows display the current scene (i.e., environment).
- The Hierarchy window is a hierarchical text representation of every GameObject in the Scene. Each item in the Scene has an entry in the hierarchy.
- The Inspector window allows you to view and edit all the properties of the currently selected GameObject.



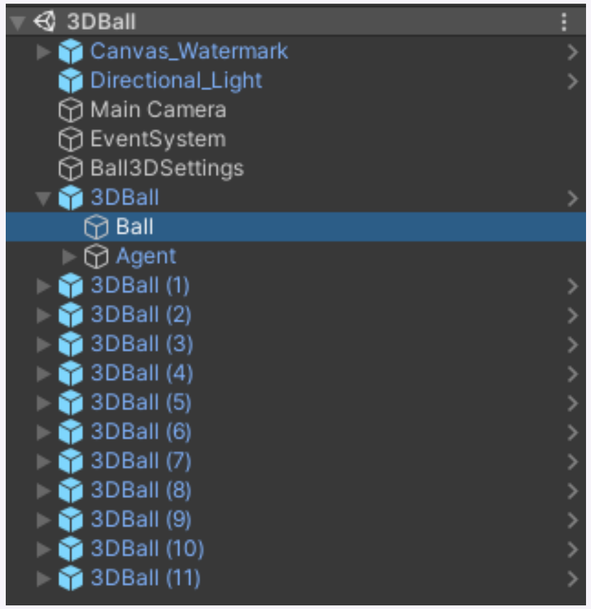
3DBall

The 3DBall environment contains a number of agent cubes and balls (which are all copies of each other). Each agent cube tries to keep its ball from falling by rotating either horizontally or vertically.



3DBall

- Open and analyze the 3DBall example from the "Project" window:
  - Go to Assets > ML-Agents > Examples > 3DBall > Scenes:
  - Open the "3DBall" scene that will appear in the "Scene" window.



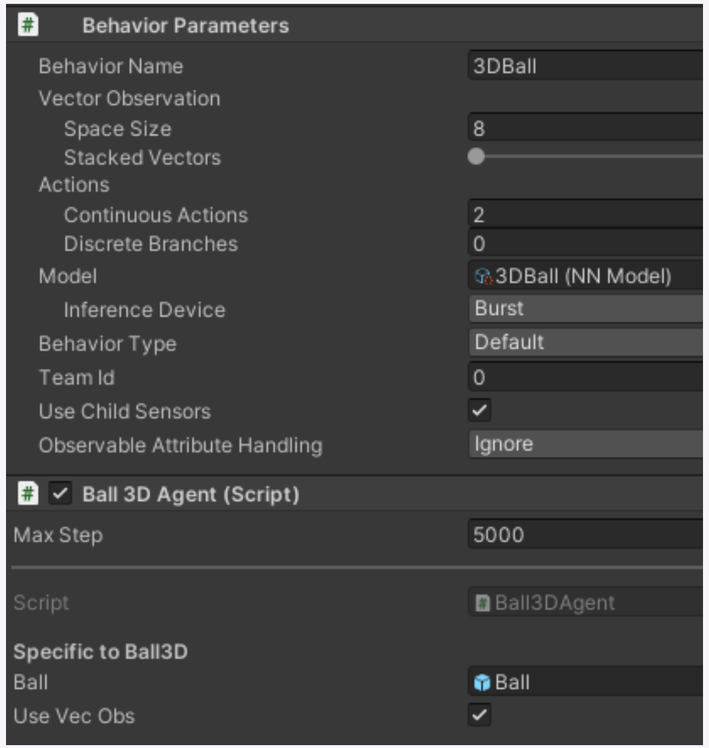
Every object in the scene is called "GameObject", which is an "abstraction" of a generic object that we can fill with different components:

- Scripts
- Colliders
- Meshes
- ....

In this scene, each 3DBall GameObject is composed by a ball (i.e., a spheric mesh with a collider) and the agent that we will train.

3DBall Agent

- Click on the Agent and analyze the "Inspector" window:

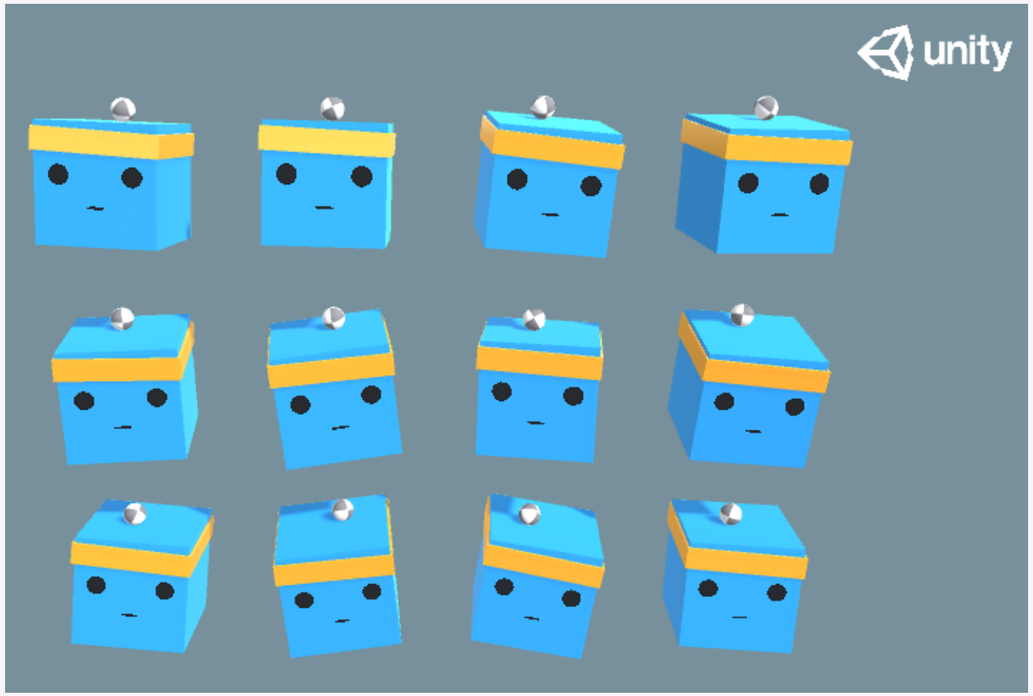


Each agent has a "Behavior Parameters" script that sets the parameters for the training (e.g., observation and action space, ...) and the max step that the agent performs in each episode.

We will later explain what "training" and its components are.

3DBall Execution

- We can notice in the inspector, that a pre-trained model is already available (in the "Model" field), so we can execute the environment and see a trained model in action.



Exercise: Train 3DBall

Instead of using the pre-trained model, we can train our own neural network to solve the task, following this link:

- In a terminal with the conda environment activated, navigate to the ML-Agents toolkit folder.
- Run "mlagents-learn config/ppo/3DBall.yaml --run-id=first3DBallRun"
- When the message "Start training by pressing the Play button in the Unity Editor" is displayed, press the Play button in Unity to start training.

This will train a policy with the policy-gradient algorithm "Proximal Policy Optimization"

Test the model

Replace the "Model" field in the inspector with the trained model, which is saved in the "results" folder in the toolkit path.

Hence, you can test the performance of this by executing the environment in the inference setting.

Material

Unfortunately, Unity is a world-leading powerful, fast, and flexible platform that is used by major game development companies (e.g., Blizzard) so we can not cover all the important aspects and fundamentals.

- For more details about Unity, we remind to the official doc
- For more details about the 3DBall environment and the toolkit, we remind to the GitHub

Exercise

Given the Unity environment available here and the training scripts in the course's GitHub (you have to launch "main.py" for the training):

- Add the perimetral walls in the environment, setting Tag (in the inspector) to "Obstacle"
- Setup the Lidar sensor (i.e., Ray Perception Sensor 3D in the inspector) to cast 11 scans in a [-90, 90] degree distribution with length 4 (remember to adjust the offset and the cast radius of the sensor)

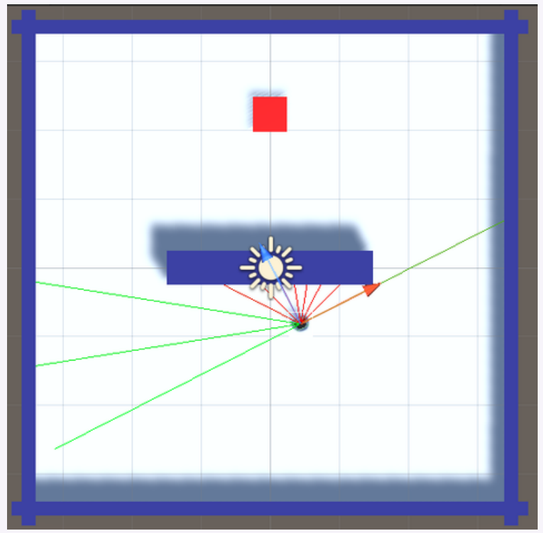
The following link contains a complete tutorial to design a Unity environment, where you can find information on how to create/add a 3D object.

Notice that you need Tensorflow 2.3 to execute the python scripts (i.e., run "pip3 install tensorflow==2.3" in the conda environment) as well as other dependencies (e.g., matplotlib) that you have to easily install with pip.

Exercise

Given the Unity environment available here and the training scripts in the course's GitHub (you will launch "main.py" for the training):

- Download the Python scripts for the training with Double DQN and train the environment without obstacles.
- Try to reproduce the Bug0 environment and train it with the same algorithm

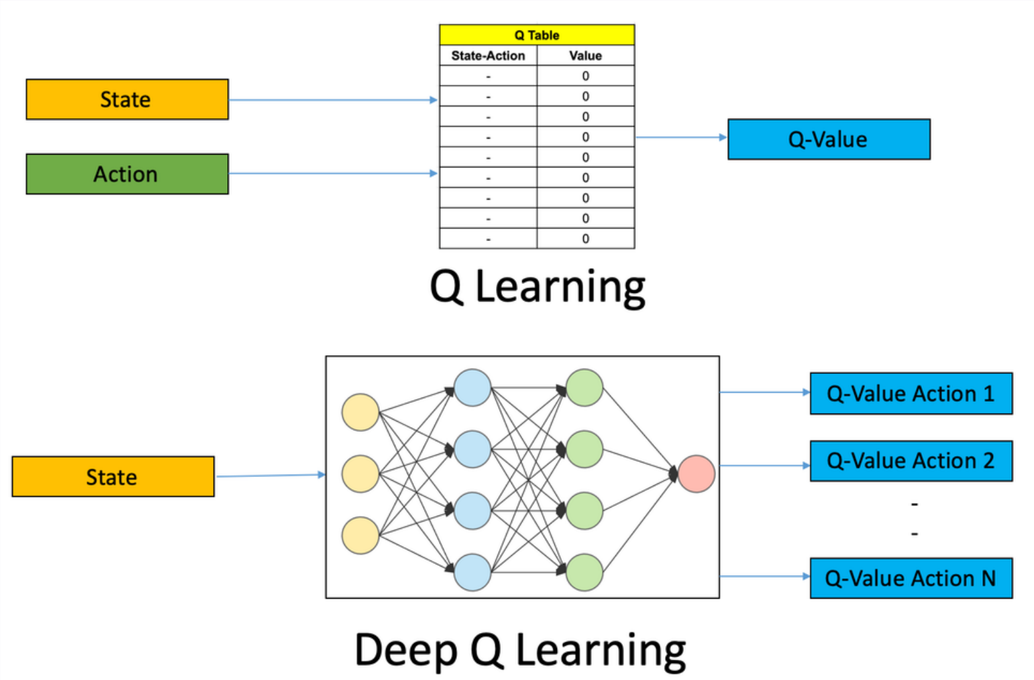


The obstacle is in position (0, 0, 0) with scale (3, 1, 0.5)

Deep Q-Learning

We assume knowledge of the classic Q-Learning algorithm that you saw in the AI course.

The only difference between Q-learning and DQN is the "agent's brain", i.e., we replace the Q-table with a Deep Neural Network.

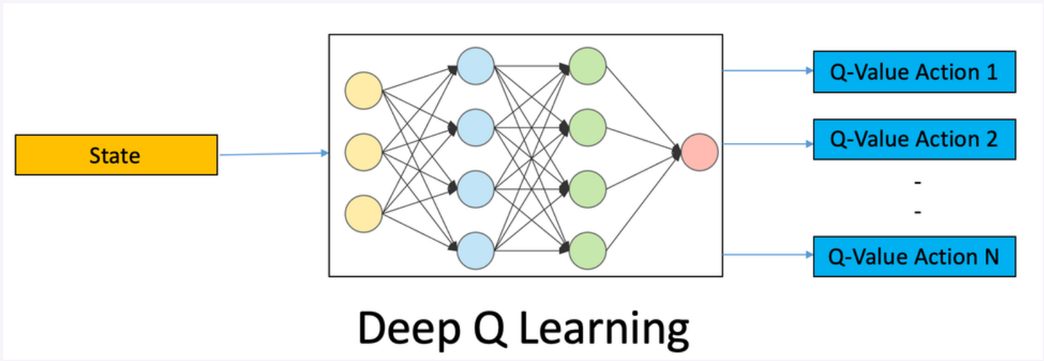


Deep Q-Learning

Hence, we feed the network with relevant information about the desired task:

- Goal's heading and distance
- Laser scans
- ....

And we map the outputs of the network (i.e., Q-values) to motor velocities (i.e., actions)



Double Deep Q-Learning

The problem with standard DQN is that the same network computes the Q-values and its update rule, hence the same approximator chooses and evaluates the performed action and this (intuitively) can cause overestimation.

Hence, we introduce an additional network that estimates the value of the performed action for the update rule of the network's weights.

However, these algorithms rely on different parameters.

Hyperparameters

- gamma (i.e., discount factor): is a value in range [0, 1) that is used to give more credit to immediate rewards (values closer to 0) or to delayed rewards (values closer to 1).
- eps_d: measures how fast to decay the exploration.
- buffer - size: is the memory that stores experiences for the training.
- buffer - batch: is the size of the sample that is used to train the network.
- dnn - h_layer, h_size: is the size of the network.

```
agent:
  update_start: 1
  polyak: True
  tg_update: 5

  gamma: 0.99
  tau: 0.005

  eps: 1
  eps_min: 0.05
  eps_d: 0.99

  buffer:
    size: 20000
    batch: 128

  dnn:
    h_layers: 2
    h_size: 64
```

Exercise

Given the Unity environment and the python scripts:

- Try to train the navigation using different hyperparameters
- Try to change the reward function inside the "TurtleAgent" script in Unity