|  |  |
| --- | --- |
| 6C5- MultiJoueur  Tutoriel et Formatif | |
| **Nom, prénom et DA** | * Nom, Prénom (1234567) |
| **Cours** | **6C5 – Applications de jeux et simulations II** |
| **Enseignant** | Julien Brunet |

Table des matières

[MLAgents 1](#_Toc134443839)

[Première Expérience d’apprentissage 2](#_Toc134443840)

[Formatif 6 5](#_Toc134443841)

[Consignes de remise du travail 5](#_Toc134443842)

[Références : 9](#_Toc134443843)

# MLAgents

Unity ML Agents est un module permettant d’entrainer des agents en Unity basé sur les principes de l’apprentissage par renforcement

Une image contenant diagramme

Description générée automatiquement

<https://larevueia.fr/apprentissage-par-renforcement/>

L’apprentissage par renforcement avec MLAgent

Vous trouverez une multitude d’exemples [ici](https://unity.com/fr/products/machine-learning-agents).

# Installation de l’environnement d’apprentissage

Une image contenant table

Description générée automatiquementMise en place de ML-Agents

Tout le processus d’installation est décrit [ici](https://github.com/Unity-Technologies/ml-agents/blob/develop/docs/Installation.md). Je vous propose ici une version simplifiée :

1. Installer python 3.10.2 (l’installer est sur TEAMS) avec toutes les options par défaut
2. Installer miniconda 3 (l’installer est sur TEAMS) avec toutes les options par défaut.
3. Telechargez sur TEAMS ml-agents-release\_22.zip et dézippez le.

C’est le projet que l’on trouve sur<https://github.com/Unity-Technologies/ml-agents/tree/develop> mais adapté à notre version d’Unity

1. Ouvrir le projet dans Unity en utilisant la version habituelle d’Unity (Nom : project, il n’est pas à la racine! **Attention**, ignorer l’erreur de compilation)
2. Ouvrir la scène : 3DBall
3. Installer les package com.unity.ml-agents et com.unity.ml-agents.extensions

(**Utiliser Add package from disk et prendre ceux inclus dans le projet cloné!)**

1. Créer un environnement virtuel python (à faire une seul fois)

Dans la fenêtre anaconda prompt (ou un autre terminal dans le répertoire miniconda), exécuter la commande :

conda create -n mlagents python=3.10.12

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

1. Pour démarrer votre environnement virtuel python, exécuter la commande ( à faire lors de chaque utilisation) :

conda activate mlagents

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

1. Installer Pytorch, exécuter la commande (peut prendre plusieurs minutes) :

pip3 install torch~=2.2.1 --index-url <https://download.pytorch.org/whl/cu121>

Une image contenant capture d’écran, texte, Police

Description générée automatiquement

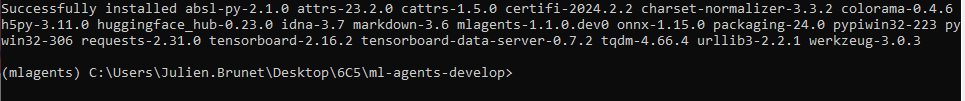
1. Install MLAgents, aller (cd) cans le répertoire ml-agents cloné :

 cd /path/to/ml-agents-release\_22

puis exécuter les commandes

python -m pip install ./ml-agents-envs

python -m pip install ./ml-agents



1. Tester en exécutant mlagents-learn --help

Une image contenant texte, logiciel, Police, Logiciel multimédia

Description générée automatiquement

1. Pour démarrer un apprentissage, commencer par aller dans un répertoire de travail (ou les résultats d’apprentissages seront déposée) et exécuter la commande :

mlagents-learn

Une image contenant texte

Description générée automatiquement

1. Starter l’exécution dans unity, l’apprentissage débute!

Une image contenant texte, Logiciel multimédia, noir, logiciel

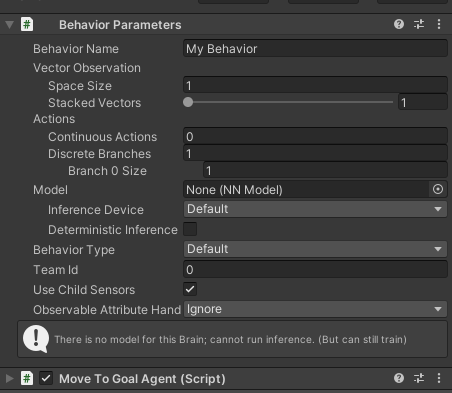
Description générée automatiquement

Observations des différents aspects de l’apprentissage!

# Première Expérience d’apprentissage

Configurons une première scène pour l’apprentissage

1. Cloner le projet 3.01-ML-agents-Formatif sur GitHub.
2. Ouvrir le projet, et importer la version des package locaux de mlagents et mlagents- extensions
3. Ouvrir la scène MoveToGoal dans MoveToGoal
4. **Créer un script** MoveToGoalAgent.cs **faisons-le hériter de Agent** (Unity.MLAgents). Ce script est celui responsable du comportement de l’agent et de ses informations pour l’apprentissage.
5. Ajoutons-le sur l’agent et observons **l’ajout automatique du composant BehaviourParameters**



1. Ajoutons encore un composant DecisionRequester.cs sur l’agent qui va permettre de prendre des décisions. Celui-ci n’a pas à être modifié, mais il ne faut pas l’oublier !!
2. Redéfinissons la méthode OnActionReceived de MoveToGoalAgent qui affiche le contenu de la variable actions.DiscreteActions.

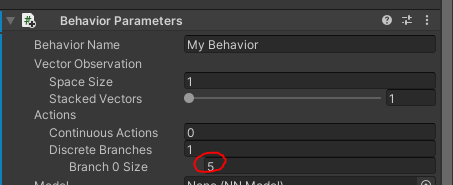
public override void OnActionReceived(ActionBuffers actions)

{

Debug.Log(actions.DiscreteActions[0]);

}

1. Configurons le Branch 0 size a 5 pour accepter des valeurs entières entre 0 et 4.



Démarrons le premier apprentissage et observons les décisions prises par l’agent

1. Démarrons MLagents-learn.exe (effacer au préalable ppo dans le dossier result ou utiliser --force):

Une image contenant texte

Description générée automatiquement

1. On appuie sur play dans unity et cela démarre l’entrainement…On peut l’interrompre. En fait une succession d’épisodes a été démarré et à chaque cycle on a une nouvelle valeur de actions.DiscreteActions[0]qui a été générée… **Ce sont les décisions** (pour le moment totalement aléatoires) **prises par l’agent.** On va maintenant interpréter les décisions comme des mouvements !!
2. On constate que cela a généré un répertoire result/ppo (dans votre répertoire courant) qui contient un fichier .onnx qui est le « cerveau généré ». Il contient les informations qui permettront ensuite à l’agent de prendre des décisions en fonction de ce qu’il aura appris. Le nom de ce fichier est le Behavior Name des behavior parameters :

Une image contenant texte, capture d’écran, Logiciel multimédia, Police

Description générée automatiquement

1. On peut observer que si on relance le MLAgents-learn on a une erreur. C’est dû au fait qu’on a déjà un fichier onnx généré. On peut soit effacer le résultat dans result, soit ajouter  **- - force** à la commande, soit préciser un autre run-id comme ici.

Une image contenant texte

Description générée automatiquement

1. **Petit exercice :** essayer de faire de même avec une Continuous Action pour observer leur utilisation (On a des Floats entre -1 et 1 au lieu d’integer)

**Bilan** : L’agent peut décider de prendre des décisions entières ou continues et leur nombre est déterminée par le programmeur. Tout se passe dans les Behavior Parameters :

Une image contenant capture d’écran, texte, Logiciel multimédia, Police

Description générée automatiquement

Donnons-lui des observations sur lesquelles il pourra prendre ses décisions:

On a ce message, du au fait que l’on a fourni aucune information en entrée à l’agent, les observations :



1. De quoi a-t-il besoin pour comprendre où il est par rapport au but? Sa position à lui et celle du but!
2. Ajoutons donc une référence targetTransform vers le transform du but dans le script de l’agent.
3. Ensuite on ajoute la méthode **suivante, qui est la base de la collecte d’informations.**

//This methods is the "eyes" of the agent

//What it collects is what it knows about

//Here: the position of the goal in the local space (x,y,z)

//its position in the local space(x, y, z)

public override void CollectObservations(VectorSensor sensor)

{

//the position of the agent in the local space (x,y,z)

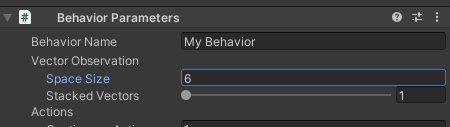
sensor.AddObservation(transform.localPosition);

//the position of the goal in the local space (x,y,z)

sensor.AddObservation(targetTransform.localPosition);

}

1. Comme on a deux vecteurs d’observation, on va mettre la space size a 6 =2x3



(Notons que stacked vectors permettrait d’aussi tenir compte des valeurs précédentes, dans des cas plus complexes.)

1. Modifions maintenant les actions pour les interpréter comme un mouvement en x et un mouvement en y

//This methods is what the agent does with the action decided

public override void OnActionReceived(ActionBuffers actions)

{

//First action is X move

float moveX = actions.ContinuousActions[0];

//second action is Z move

float moveY = actions.ContinuousActions[1];

//should be a member variable

float speed = 10;

//Move!!!

transform.Translate(new UnityEngine.Vector3(moveX, 0, moveY) \* Time.deltaTime \* speed);

}

1. De la même manière qu’avant on configure donc 2 continuous actions :



Comme dit précédemment elles symbolisent les décisions prises par l’agent à nous de savoir comment les interpréter!

**Bilan** : Il faut choisir soigneusement les observations que l’on donne à l’agent qui lui permettront d’Apprendre le plus vite possible (sans perdre la généricité de la situation)!

Il est temps maintenant de récompenser notre agents (ou de le punir!):

1. Ajoutons un trigger sur l’atteinte du but qui donne une récompense à l’agent (AddReward(1f)) et qui termine l’épisode
2. Ajoutons un trigger sur la collision avec les murs qui donne une punition (récompense négative) à l’agent et qui termine l’épisode.
3. Ajoutons un reset du transform au début de l’épisode

**Très, très important : Nous allons dupliquer les scènes pour améliorer l’entrainement, il est donc capital d’utiliser des positions locales**

//This is how reward is managed

private void OnTriggerEnter(Collider other)

{

//On collision with goal... success!!!

if (other.CompareTag("goal"))

{

AddReward (1f);

EndEpisode();

}

//On collision with wall... defeat!!!

else if (other.CompareTag("wall"))

{

AddReward (-1f);

EndEpisode();

}

}

//This is what happens on episode start

public override void OnEpisodeBegin()

{

transform.localPosition = new Vector3(-1, 1, 0);

}

**Bilan** :

* Il faut absolument utiliser des positions locales (relatives au parent sinon on aura des comportements étranges.
* On peut forcer la fin de l’épisode (ou le laisser atteindre son MaxSteps)
* Le reward peut être positif ou négatif et peut être fait à n’importe quel moment du code (Update() ou dans un trigger par exemple)
* OnEpisodeBegin permet de mettre les choses en position initiales

Ajoutons une heuristique juste pour tester :

1. On ajoute une méthode (public override void Heuristic(in ActionBuffers actionsOut)) pour tester notre agent manuellement. Elle n’a pas d’autre vocations que de simuler un entrainement avant de le réaliser

//This is how to move the agent when keyboard driven...

//Behaviour Type = Heuristic Only or Default with no brain and no training...

public override void Heuristic(in ActionBuffers actionsOut)

{

ActionSegment<float> contActions = actionsOut.ContinuousActions;

contActions[0] = Input.GetAxisRaw("Horizontal");

contActions[1] = Input.GetAxisRaw("Vertical");

}

1. Testons le mouvement manuel de l’Agent (sans apprentissage).
2. Ajoutons des couleurs pour voir ce qu’il se passe (vert si succès, rouge si échec)

//two materials to color the scene in case of success or failure to give a visual feedback

[SerializeField] private Material succesMaterial;

[SerializeField] private Material failureMaterial;

[SerializeField] private Renderer floorRenderer;

//This is how reward is managed

private void OnTriggerEnter(Collider other)

{

//On collision with goal... success!!!

if (other.CompareTag("goal"))

{

AddReward (1f);

floorRenderer.material = succesMaterial;

EndEpisode();

}

//On collision with wall... defeat!!!

else if (other.CompareTag("wall"))

{

AddReward (-1f);

floorRenderer.material = failureMaterial;

EndEpisode();

}

}

**Bilan** : L’heuristique ne sert à rien d’autre que tester votre code mais je vous conseille de le faire très soigneusement !!!!

Commençons l’entrainement

1. Dupliquer la scène une vingtaine de fois (Je vous conseille de faire d’abord un prefab pour vos modifications futures!
2. Lancer le module python et play…. Le vrai entrainement débute. On le laisse se terminer si possible (ou on l’écourte). Il crée un fichier onnx que l’on peut assigner à notre agent!
3. Il n’y a plus qu’à tester en mode Inférence only (une scène unique, on peut cacher les autres)

Réfléchissons aux améliorations…

Que penser du résultat? Quel est le problème… c’est maintenant qu’il faut bien réfléchir…

Modifier pour pénaliser chaque step! Le but est de le forcer à prendre des risques!

1. Ajouter une récompense négative à chaque pas !

//Pour minimiser la durée..

AddReward(-0.01f);

1. Recommencer l’entrainement!

Modifier pour avoir une position random au but et à l’agent.

1. Si on déplace le but ça ne marche plus!! Normal on ne l’a pas entrainé!
2. Mettre une position aléatoire au but et à l’agent au début de l’épisode et réentraîner!

//This is what happens on episode start

//Just put the agent on random position

//Just put the goal on random position

public override void OnEpisodeBegin()

{

transform.localPosition = new Vector3(-1, 1, 0);

Vector3 move = new Vector3(Random.Range(-3f, 1f), 0, Random.Range(-2f, 2f));

transform.Translate(move);

targetTransform.localPosition = new Vector3(0, 0, 0);

move = new Vector3(Random.Range(1f, 3f), 0, Random.Range(-2f, 2f));

targetTransform.Translate(move);

}

1. Tester!

Utiliser le tensorboard

Tensorboard :

1. Dans un autre cmd, aller dans votre répertoire de travail
2. Et taper les commandes :

conda activate mlagents

tensorboard --logdir results

1. Ouvrir l’url : http://localhost:6006/

Configurer l’entrainement

Tout est ici :

<https://unity-technologies.github.io/ml-agents/Training-ML-Agents/#training-with-mlagents-learn>

L’utilisation d’un run-id permet de continuer à entrainer des apprentissages spécifiques :

>mlagents-learn --initilize-from=oldRunId --run-id=newRunId

# ~~Formatif 6~~

**~~Option 1~~**

~~Modifier ce qui précède pour ajouter un obstacle dans la scène positionné aléatoirement.~~

~~L’agent doit atteindre le but, sans toucher l’obstacle.~~

**~~Option 2~~**

~~Modifier ce qui précède pour avoir 2 buts successifs dans la scène positionnés aléatoirement.~~

~~L’agent doit atteindre un but précis, puis l’autre dans l’ordre avant d’avoir terminé!~~

~~Commencez par r~~**~~éfléchir aux informations nécessaires à l’agent~~** ~~c’est le plus important!~~

~~Ensuite configurez votre nouvelle scène avec les deux buts et~~ **~~testez-la avec l’heuristique~~**~~.~~

~~Une fois que tout fonctionne essayez d’entrainer avec une scène d’abord puis augmenter le nombre de scènes… Tester votre cerveau!~~

Scripts :

MoveToGoalAgent.cs

|  |
| --- |
| using Unity.MLAgents;  using Unity.MLAgents.Actuators;  using Unity.MLAgents.Sensors;  using UnityEngine;  public class MoveToGoalAgent : Agent  {  //The transform of the goal to reach  [SerializeField] private Transform targetTransform;  //two materials to color the scene in case of success or failure to give a visual feedback  [SerializeField] private Material succesMaterial;  [SerializeField] private Material failureMaterial;  [SerializeField] private Renderer floorRenderer;  //Agent speed for configuration purpose  [SerializeField] private float speed = 6;  //This methods is what the agent does with the action decided  public override void OnActionReceived(ActionBuffers actions)  {  //First action is X move  float moveX = actions.ContinuousActions[0];  //second action is Z move  float moveY = actions.ContinuousActions[1];  //Pour minimiser la durée..  AddReward(-0.01f);  //Move!!!  transform.Translate(new UnityEngine.Vector3(moveX, 0, moveY) \* Time.deltaTime \* speed);  }  //This methods is the "eyes" of the agent  //What it collects is what it knows about  //Here: the position of the goal in the local space (x,y,z)  //its position in the local space(x, y, z)  public override void CollectObservations(VectorSensor sensor)  {  //the position of the agent in the local space (x,y,z)  sensor.AddObservation(transform.localPosition);  //the position of the goal in the local space (x,y,z)  sensor.AddObservation(targetTransform.localPosition);  }  //This is how reward is managed  private void OnTriggerEnter(Collider other)  {  //On collision with goal... success!!!  if (other.CompareTag("goal"))  {  AddReward (1f);  floorRenderer.material = succesMaterial;  EndEpisode();  }  //On collision with wall... defeat!!!  else if (other.CompareTag("wall"))  {  AddReward (-1f);  floorRenderer.material = failureMaterial;  EndEpisode();  }  }  //This is what happens on episode start  //Just put the agent on random position  //Just put the goal on random position  public override void OnEpisodeBegin()  {  transform.localPosition = new Vector3(-1, 1, 0);  Vector3 move = new Vector3(Random.Range(-3f, 1f), 0, Random.Range(-2f, 2f));  transform.Translate(move);  targetTransform.localPosition = new Vector3(0, 0, 0);  move = new Vector3(Random.Range(1f, 3f), 0, Random.Range(-2f, 2f));  targetTransform.Translate(move);  }  //This is how to move the agent when keyboard driven...  //Behaviour Type = Heuristic Only or Default with no brain and no training...  public override void Heuristic(in ActionBuffers actionsOut)  {  ActionSegment<float> contActions = actionsOut.ContinuousActions;  contActions[0] = Input.GetAxisRaw("Horizontal");  contActions[1] = Input.GetAxisRaw("Vertical");  }  } |

# Références :

Starting with MLAgents :

<https://www.youtube.com/watch?v=zPFU30tbyKs>

ML-Agents GitHub

<https://github.com/Unity-Technologies/ml-agents>

ML-Agent référence

<https://docs.unity3d.com/Packages/com.unity.ml-agents@2.0/api/Unity.MLAgents.html>

Pour aller plus loin, … imitation learning…

<https://www.youtube.com/watch?v=supqT7kqpEI>