3D 互動數位內容製作

期末報告

Unity ML-Agent AI Car

1053320 黃莉婷

起初因為一直試不出來課堂上的方法，為了重振信心，因此到 Unity 官方提供的教程一步一步跟著做，希望能先有個好兆頭。

首先是環境設定，完全參照此網站

https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Installation-Windows.md

Unity 和 Anaconda 之前已經裝好了，主要是要新創一個 Pythob 環境，安裝

- Python3.6
- Tensorflow1.7.1

並使用官方範例(ml-agent 專案內的 3D-Ball)，將之修改成我要的 AI 車子。

開始修改之前，要先看懂原本的內容

如何**放入已經訓練好的模型**和如何**自己訓練出一個模型**看這

https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Basic-Guide.md

3D-Ball 的**程式碼解說**看這

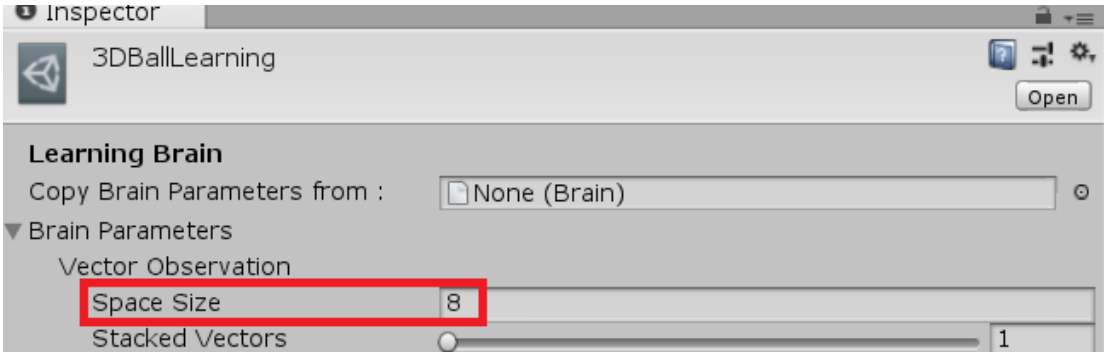
https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Getting-Started-with-Balance-Ball.md

也有中文翻譯好的網站 https://blog.csdn.net/u010019717/article/details/80472011

針對 Ball3DAgent.cs 程式碼解說，官網並沒有講得很詳細，以下會有些許我自己理解的補充，若有錯誤理解還請告知。

Ball3DAgent.cs 內有三個 function

- CollectObservations()
  - 負責收集 Agent 對環境的觀察數據，範例中觀察平台的旋轉角度、球的位置等共 8 個，所以會調用 8 次 AddVectorObs(position 內涵 x,y,z 三個變數，因此算三個)，會產生大小為 8 的空間給 Brain 計算。
  - 此數必須要與 Brain(3DBallLearning)的向量空間觀察大小相同



- AgentAction(float[] vectorAction , string textAction)
  - 接收 Brain 選擇的步驟，此範例環境為連續的運動空間類型(亦有離散的)，多數範例會在這裡分配獎勵
- AgentReset()
  - Agent 重置時會調用(範例中重置平台角度和球的位置等)，適當使用 random(隨機調整數值)可以使測資多元，較容易訓練出不同情況下的反應

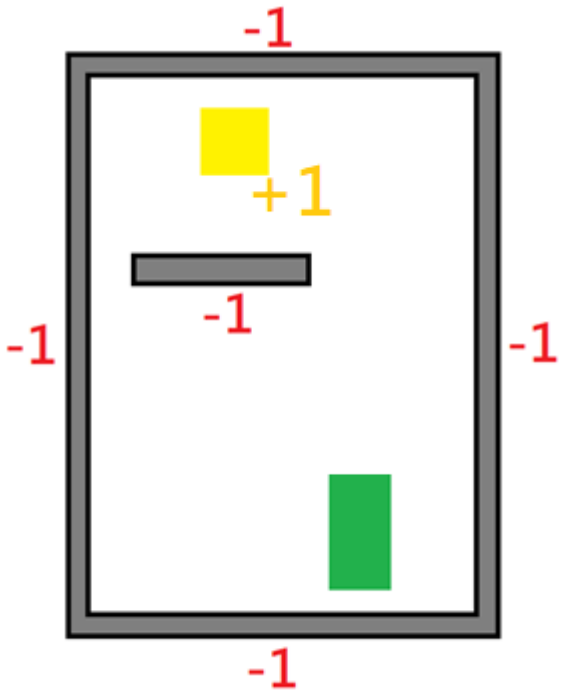
看懂原本的程式碼在寫甚麼之後，我們就來改改看。首先，整理一下要修改的部分

	[原本範例]	>>	[更換成]
●	球	>>	方塊 (車子)
●	控制平台	>>	控制車子
●	控制 X,Z 軸旋轉方向	>>	控制車子位移、旋轉
●	[獎懲] 沒掉+0.1，掉了-1	>>	底達目標+1、撞到障礙物-1
●	觀察 xxxxxx	>>	觀察車旋轉、和目標&障礙物的距離

新增

- 為車子加裝前後射線，感知障礙物和目標

概念設計圖 ：



以下為實際修改過後的程式碼:

Using 和原本的一樣，不變

```
1    using System.Collections;
2    using System.Collections.Generic;
3    using UnityEngine;
4    using MLAgents;
```

新增變數

```
18  public class Ball3DAgent : Agent
19  {
20      //[Header("Specific to Ball3D")]
21      //public GameObject ball; //改成car
22      public GameObject car;
23
24      //new
25      public GameObject goal;
26      public GameObject block;
27      public GameObject plane;
28
29      //new-車子前後射線相關
30      GameObject hitObject, re_hitObject;
31      float rayDistance = 100.0f;
32      float goal_dis;
33      float wall_dis;
34      float re_goal_dis;
35      float re_wall_dis;
36
37      //private Rigidbody ballRb;//用不到
```

註解掉 InitializeAgent()，用不到

```
38      /*  //用不到
39      public override void InitializeAgent()
40      {
41          ballRb = car.GetComponent<Rigidbody>();
42      }
43      */
```

清空改寫 CollectObserations()，此時觀察對象有 9 個

```
58      public override void CollectObservations()
59      {
60          //車子的方向
61          AddVectorObs(car.transform.rotation.z);
62          AddVectorObs(car.transform.rotation.x);
63
64          //觀察車和goal的距離
65          AddVectorObs(car.transform.position - goal.transform.position);
66
67          //觀察車和障礙物的距離
68          AddVectorObs(wall_dis);
69          AddVectorObs(goal_dis);
70          AddVectorObs(re_wall_dis);
71          AddVectorObs(re_goal_dis);
72      }
```

新增 FixedUpdate()，時時刻刻更新前後物體距離數值

```
82      private void FixedUpdate()
83      {
84          //用本物件的位置及方向產生射線
85          Ray ray = new Ray(transform.position, transform.forward);
86          //反方向射線
87          Ray ray_reverse = new Ray(transform.position, -transform.forward);
88          RaycastHit hit,re_hit;  //re = reverse
89
90          goal_dis = rayDistance;
91          wall_dis = rayDistance;
92          re_goal_dis = rayDistance;
93          re_wall_dis = rayDistance;
94          //前方射線
95          if (Physics.Raycast(ray, out hit, rayDistance))
96          {
97              hitObject = hit.transform.gameObject;
98
99              if (hitObject.tag == "goal")
100                 goal_dis = Vector3.Distance(hit.point, transform.position);
101             else if(hitObject.tag == "wall")
102                 wall_dis = Vector3.Distance(hit.point, transform.position);
103         }
104         //後方射線
105         if (Physics.Raycast(ray_reverse, out re_hit, rayDistance))
106         {
107             re_hitObject = re_hit.transform.gameObject;
108
109             if (hitObject.tag == "goal")
110                 re_goal_dis = Vector3.Distance(re_hit.point, transform.position);
111             else if(hitObject.tag == "wall")
112                 re_wall_dis = Vector3.Distance(re_hit.point, transform.position);
113         }
114     }
```

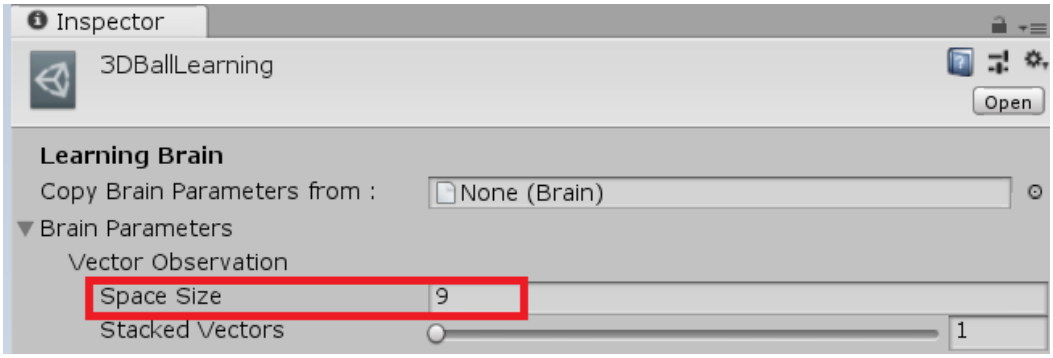
更改 AgentAction()

```
126     public override void AgentAction(float[] moveAction, string textAction)
127     {
128
129         if (brain.brainParameters.vectorActionSpaceType == SpaceType.continuous)
130         {
131             //moveAction[0]---左轉
132             //moveAction[1]---右轉
133             moveAction[0] = Mathf.Clamp(moveAction[0], -1f, 1f);
134             moveAction[1] = Mathf.Clamp(moveAction[1], -1f, 1f);
135
136             //Debug.Log(moveAction[0]+ "**********" + moveAction[1]);
137
138             car.transform.Rotate(new Vector3(0, 10 * moveAction[0], 0));
139             car.transform.Translate(0, 0, moveAction[1]);
140         }
141     }
```

清空改寫 AgentReset()

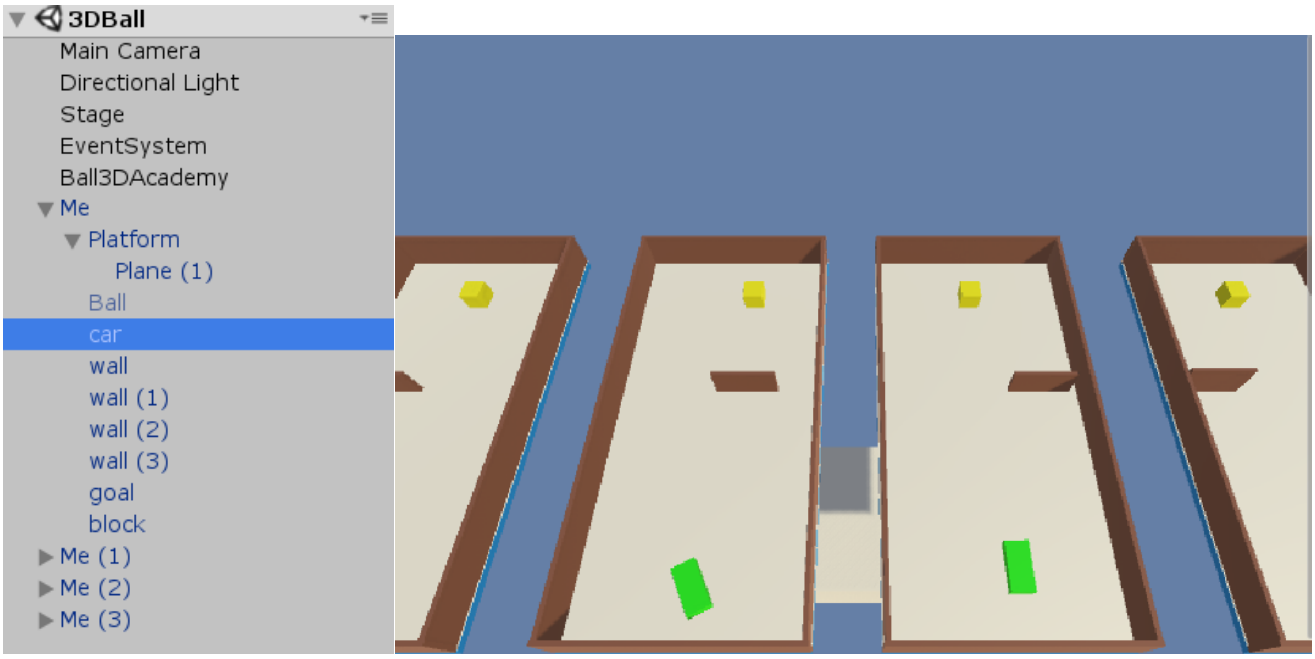
```
208     public override void AgentReset()
209     {
210         car.transform.position = new Vector3(Random.Range(-3f, 3f), 0.5f, -5f) + plane.transform.position;
211         block.transform.position = new Vector3(Random.Range(-3f, 3f), 0.5f, 1.5f) + plane.transform.position;
212         goal.transform.position = new Vector3(Random.Range(-3f, 3f), 0.5f, Random.Range(-3f, 6f)) + plane.transform.position;
213 
214         car.transform.Rotate(0,0,0);
215     }
```

因為在 CollectObserations()，觀察對象有 9 個，所以這裡數目也要改



以上是程式碼的部分。

完成後將此檔案(原本在 plane 物件下)改放到 Car(自己新增的 cube)物件下

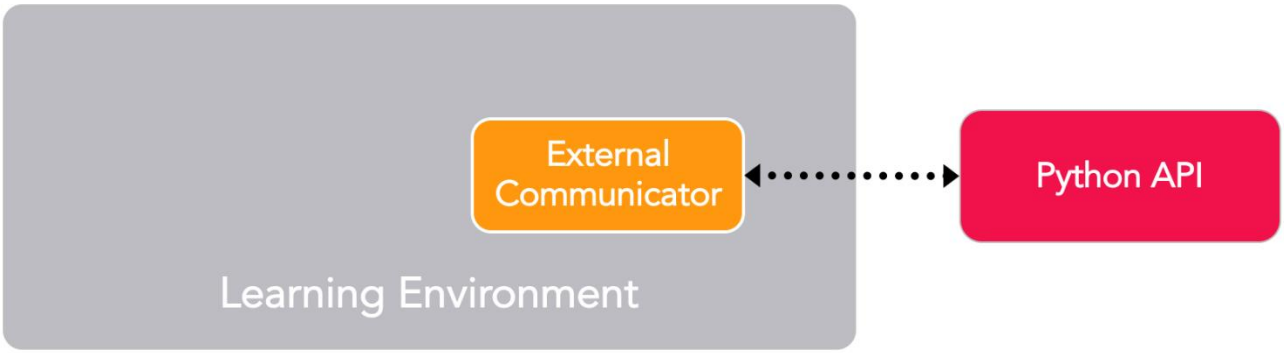


最後照著前面訓練模型的方式即可。

ML-Agents 概述

ML-Agents 是一個 Unity 插件，包含了三個高級組件

- 學習環境 – 包含 Unity 場景和所有遊戲腳色
- Python API – 包含用於訓練的所有機器學習算法，與學習環境不同的是，Python API 不適 Unity 的一部分，而是位於外部通過 External Communicator 與 Unity 進行通訊
- External Communicator - 他將 Unity 環境與 Python API 串接起來。位於 Unity 環境中



機器學習過程

在 3D-Ball 範例中使用強化學習(Reinforcement Learning)訓練 Brain。為了訓練 agent 對球進行平衡，使用一種稱為 Proximal Policy Optimization(PPO)的強化學習算法。與其他許多 RL 算法相比，經證明這種算法是更有效、通用的方法。

為了在 Ball Balance 環境中訓練 Agent，將使用 Python 包。官方提供了一個方便的命令 mlagents-learn，用於接收訓練等參數。

總結來說，實際上的操作就是開啟 cmd 到 ml-agents 資料夾下，輸入

```
mlagents-learn config/trainer_config.yaml --run-id=<run-identifier> --train
```

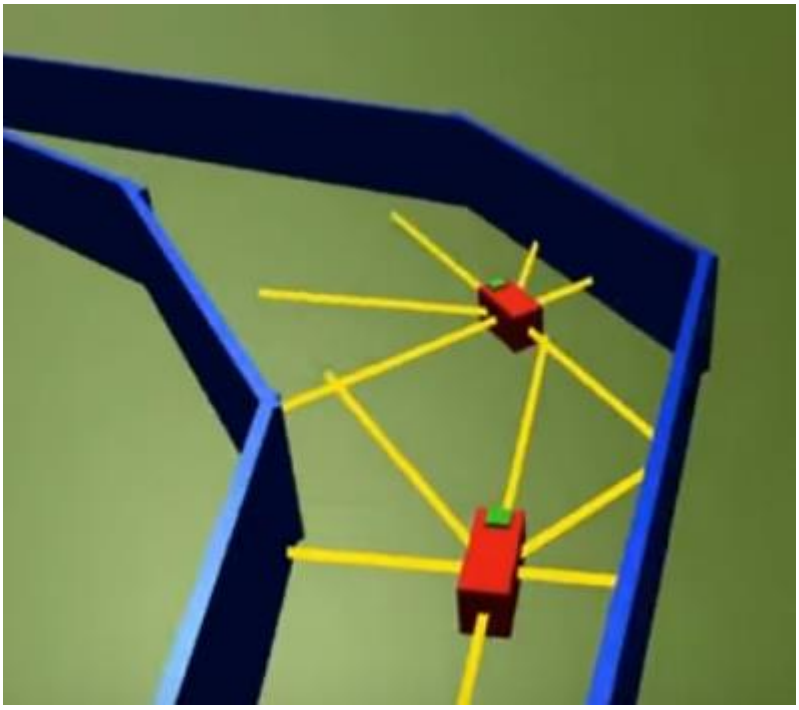
範例輸入

```
mlagents-learn config/trainer_config.yaml --run-id=firstRun --train
```

其中 YAML 是用來表達資料序列的格式。YAML 參考了其他多種語言，包括：C 語言、Python…等

後記

最後車子雖然在訓練過程中有接近 30%會抵達目標，但實際上拿訓練好的模型來跑的話，會經常在原地猶豫不決，因此後續考慮在車子上多裝幾個射線(如下圖)，協助判斷。



果然要在一天內看完所有 ML-Agnet 的觀念實在有點吃不消，因此我打算日後慢慢看完所有 github 上 ml-agnet 這專案的文件，由於我英文也不太好，所以我已經將此專案 fork 一份到我的 github 上，打算每天看一點，並在看過後將之翻譯成繁體中文，讓英文不好的人也可以輕鬆學機器學習。

我的 github: https://github.com/tsumikihuang/ml-agents-/blob/master/README.md

我完成的作品: https://drive.google.com/open?id=1xX9QtmwWfczfuY9_ImD7bP-H32KwHIVb