Versió 1

Primera fase

1. Representació del model NPC dintre de Unity

El NPC l’he representat com un objecte que com a fills te *AnchorLeft, AnchorRight and Agent*.

Els Anchors son objectes buits que tenen un component de detecció de Collisions. El seus comportaments no son de collisió típica ja que realmente l’Agent mai farà collisió real sobre els Anchors, però tindran la propietat de ser triggers d´events.

Unity té una API que permet detectar aquests events a nivel de C#.

L’Agent es un objecte que també te un detector de col·lisions, i com a fills uns objectes que permeten projectar rajos per detectar on esta la col·lisió més pròxima. La col·lisió més pròxima dependrà de com s’han configurats els rajos. En aquest cas surt un raig per l’esquerra i un per la dreta.

Es important notar que, l’objecte NPC no es visible en la imatge següent, però fa de contenidor. Això es important perquè l’únic objecte que es mou es el fill Agent del NPC, que es qui al final tindrà el model de Neuronal Network.

Escala de tiempo

Descripción generada automáticamente con confianza media

Texto

Descripción generada automáticamente

1. Com modelitzar un problema fent ús de Unity MLAgents

Primer de tot ens em de instal·lar el paquet ML Agents dintre d’un projecte de Unity. Per instal·lar un paquet s’ha d’anar al *Package Manager*.  
  
Jo estic treballant amb la versió d’Unity 2021.2.0f1 I amb la versió 2.0.0 de ML Agents. Es important les versions amb les que treballem perquè pot provocar problemes de compatibilitat. Al repository de [ML Agents](https://github.com/Unity-Technologies/ml-agents) d’Unity en l’apartat de *Releases and Documentation* hi ha la taula de compatibilitats entre Unity i la dependencia.

La resta d’eines que s’utilitzen per poder connectar Unity amb el procés de Deep Learning esta especificat en la següent [documentació](https://github.com/Unity-Technologies/ml-agents/blob/release_17_docs/docs/Installation.md).

Jo he fet us de l’eina *venv* de *python* per tenir installat el paquet *mlagents-learn*, aquest es un paquet fet en python.

Important:

* El escalat de temps de mlagents-learn per defecte es de x20, això pot ser configurat amb l’opció –time-scale.
* Es poden tenir diferents fitxers de configuració, per cridar-los quan comença el procés crida la instrucción *mlagents-learn {FileConfigPath}.yaml –run-id {ID}.*

1. Modelització de moure l´Agent d’esquerra a dreta cíclicament.

Per poder modelitzar el problema es necessari que creen una classe que contindrà la part lògica de la modelització i que l’estenguem de la classe *Agent* de ML Agent.

La classe *Agent* conte varis mètodes que poden ser sobreescrits en la nostra clase principal.

Mètodes que s’han sobreescrit de la classe *Agents*

* Initialize
* OnEpisodeBegin
* OnActionReceived
* CollectObservations
* Heuristic

A continuació explicaré per quina funcionalitat estan pensades aquests mètodes sobreescrits.

**Initialize**

Aquest mètode es crida un cop únicament en el cicle de vida del aprenentatge.

Esta pensat per fer *caching* d’objectes de la escena o inicialitzar valors.

Texto

Descripción generada automáticamente

**OnEpisodeBegin**

Aquest mètode es crida cada cop que un episodi d’aprenentatge ha acabat, forçadament o perquè el màxim nombre de passos del episodi s’han completat.

S’ha de ser consistent amb aquest mètode, es a dir, en el meu cas l’Agent sempre sortirà del centre del NPC, i jo forço la seva recol·locació al centre, també s’ha de tenir en compte els problemes d’inèrcia que porta l’objecte, o la re calculació de target o metes pel nou episodi.

Texto

Descripción generada automáticamente

**OnActionReceived**

Aquest mètode porta un paràmetre d’entrada, anomenat *ActionBuffer*, l’*ActionBuffer,* es un objecte que te dos arrays, un per poder representar valors continus i altres per representar valors discrets.

Pel meu problema necessitava emular el moviment lateral. Els valors que hi ha dintre de cada posició dels arrays es simbòlic, i ho has de saber tu com a desenrotllador del model.

Com he dit, només estava interessat en la emulació del moviment lateral. El rang de valors que m’espero va de -1 a 1, -1 representa moviments cap a l’esquerra i 1 cap a la dreta.

Amb aquesta informació jo puc canviar la direcció de la velocitat del Agent.

Aquesta funció pot ser cridada pel motor de Deep Learning o per interacció directa amb un jugador. En cas que el programa estigui corrent en mode d’aprenentatge, serà el motor de DeepLearning que li doni valors a *actions.ContinuousActions[0]*. En cas que no estigui aprenent, es a dir corrent en mode *Heuristic*, els valors seran donats gracies a la interacció amb la funció *Heuristic* que veurem més endavant.

En cas que estigui corrent en mode d’aprenentatge, el nombre d’elements dels arrays de *ActionsBuffer* s’han de configurar per fora, ho veurem més tard quan parlem sobre la configuració en l’entorn d’Unity.

Texto

Descripción generada automáticamente

**CollectObservations**

*CollectObservations* es un mètode que s’executa cada x temps, aquest temps d’execució es configura amb l’eina de Unity.

Fixat que igual que en el mètode *OnActionReceived* aquest te un paràmetre. En aquest cas no esta pensat per consumir les dades d’aquest paràmetres sinó per donar informació al motor de DeepLearning.

Fent ús de la referencia sensor podem fer que cada x temps, el motor de DeepLearning sàpiga dades rellevants per que ell pugui resoldre el problema.

Lo que vaig pensar que seria interesant que observes, es, la orientació del Agent respecte el target, la distancia del Agent al target, la posició del target i la velocitat amb la que el Agent es mou.

Es important notar que encara que enregistra una observació d’un objecte de tipus Vector2D, no es guarda com una observació sinó com a dos. Això es important perquè de la mateixa manera que he explicat abans del *ActionBuffer* s’aplica al *VectorSensor.* Hem de saber quantes observacions es fan realment perquè desprès amb Unity em de configura l’allargada del vector d’observacions.

**Texto

Descripción generada automáticamente**

**Heuristic**

El mètode d’*Heuristic* permet interactuar amb el jugador. Fixat que te la mateixa estructura de dades com a paràmetre, el *ActionBuffers*.  
Com he explicat prèviament, el mètode *OnActionReceived* interactúa tant amb el motor de DeepLearning com amb el jugador, i amb el jugador s’interactua mitjan aquest mètode.

Les modificacions sobre l’estructura *ActionsBuffers* de Heuristics serán rebudes en el método *OnActionReceived.*

Com només em fixo en els moviments laterals, en la primera posició del *continuos actions* ha de a tenir un valor entre -1 a 1, representant els moviments laterals.

**Texto

Descripción generada automáticamente**

1. Asdf

Imagen que contiene edificio, gabinete, puerta, hombre

Descripción generada automáticamente

1. *Veure estadístiques del procés de DeepLearning en temps real amb TensorBoard*

A aquestes altures ja em entrenat el nostre model. Al moment d’instalar el paquet de mlagents-learn de python, bé el paquet de *TensorBoard.* Aquest paquets ens permetra veura grafiques en temps real de parametres com el *Learning Rate, Cumulative Reward, Policy Loss* i altres.

Per que s’obri un servidor web i consumir les grafiques per un navegador cal anar al directori de results i acceder a la carpeta que te com a nom l’ID de l’execució del mlagents-learn.

Comanda: tensorboard --logdir .\npc01\.

Pantalla de un computador

Descripción generada automáticamente con confianza media

*Cumalitive Reward and Episode Length*

La primera gràfica representa una mitja de lo que es guanya per episodi. Te sentit ja que a mesura que el model va aprenent, el guany acumalat per episodi ha de creixer, i si ens fixem comença a estancar-se cap al final.

La segona grafica representa quan dura un episodi de mitja. També té la seva lógica ja que dintre del model de C#, en el moment que el NPC traspasa els seus limits, es reinicia de forma forçada el cicle de aprenentatge. A mesura que el model apren a no sortir-se del rang, el temps d’episodi s’aproximarà al nombre màxim de pasos configurats per episodi.

Gráfico, Gráfico de líneas

Descripción generada automáticamente

La grafica de *Policy Loss* representa quant la política canvia, un model correcte en teoría tindria que fer que aquests canvi minimitzi.

La grafica de *Value Loss* representa la mitja de perduas, un model correcte ha d’anar pujant aquesta mitja fins que sigui estable, desprès tendirà a baixar.

Gráfico, Gráfico de líneas

Descripción generada automáticamente

La grafica d’entropia representa que tan aleatori son els moviments, un model correcte ha de baixar aquesta aleaotoritat i fer més lo que ha estat aprenent.

Gráfico, Gráfico de líneas

Descripción generada automáticamente

La grafica de *Learning Rate* ens diu que tan va aprenent, a mesura que passen els passos, com es normal, a mesura que pasa el temps, aprendre noves coses per episodi sera més difici que al principi.