

Aprenentatge per reforç amb Unity

TAIA Project

Wilber Eduardo Bermeo Quito

12 de novembre de 2021

Índex

1	Problema a resoldre en una frase	3
2	Introducció i Motivació	3
3	Dades/Coneixements dels que es disposa	3
4	Proposta	4
4.1	Agent	4
4.1.1	Como modelitzar un Agent?	4
4.2	Polítiques a aprendre	5
4.2.1	Autonomia en el moviment	5
4.2.2	Saber atacar	5
4.3	Arquitectura, Algorismes, Propostes existents	6
4.3.1	Espai d'aprenentatge	7
4.3.2	Agent	7
4.3.3	Behaviour	8
4.3.4	Requester	10
4.3.5	Tensorboard	11
5	Experiments	13
5.1	Representació del objecte NPC dintre d'Unity	13
5.2	Autonomia en el moviment	14
5.2.1	Versió (I)	14
5.3	Metodologia i Resultats	18
5.3.1	Autonomia en el moviment, versió (I)	18
6	Conclusions i línies futures	19
7	Referències	19

1 Problema a resoldre en una frase

Fer que un NPC segueixi una política en concret fent us de Deep Reinforcement Learning en Unity.

2 Introducció i Motivació

Durant l'estiu del 2021 m'he estat barallant amb Unity (motor de videojocs) perquè vull fer un joc com a treball final de grau.

El joc, resumidament, serà un Metroidvania 2D, en el qual hi haurà una tira de diferents NPCs. Molts d'aquests NPCs seran personatges actius i enemics.

La motivació es clara, aprendre més d'Unity i expandir la riquesa del TFG amb les eines pròpies d'Unity que dona per implementar intel·ligència artificial als videojocs.

3 Dades/Coneixements dels que es disposa

Degüt a la naturalesa del problema seleccionat, no es necessiten conjunts de dades per poder treballar.

Coneixo l'entorn de treball Unity, no a nivell professional, però sí tinc un base.

Lo que hem permetrà encarar el problema es l'eina *ML-Agents Toolkit*. Dona totes les eines necessàries per utilitzar Unity com a motor de simulació per que els Agents de les escenes aprenguin polítiques segons una modelització del problema a resoldre.

4 Proposta

4.1 Agent

Classe que conte mètodes que poden ser sobre escrits. La seva API contempla mètodes per poder generar observacions del medi, per prendre accions i per assignar recompenses.

La classe Agent conte altres mètodes que poden ser sobre escrits. Cada Agent esta relacionat amb un Behavior Parameter class i a un Decision Requester.

4.1.1 Como modelitzar un Agent?

Es necessiten definir tres tipus d'entitats per cada moment en el joc.

- Observacions
- Accions
- Recompenses

4.1.1.1 Observacions

Les observacions poden ser numèriques i/o visuals. Les observacions numèriques són del punt de vista del observador (NPC), les visuals en canvi, són generades per col·lisions de rajos que es poden ajuntada al agent i representen el que l'agent esta veient en aquell mateix moment.

4.1.1.2 Accions

Lo que realment l'agent pot fer dintre de l'escena. Les accions poden ser representades per valors continuus o discrets.

4.1.1.3 Recompenses

Es un escalar que representa que tan bé ho esta fent l'agent. Les recompenses (negatives o positives) no tenen perquè ser donades a cada moment, es poden donar en certs moments provocats per esdeveniments. Es important saber quan i quant es recompensa o es castiga a l'agent pel seus actes. Depenenet de la política de recompenses farà que l'aprenentatge de una certa tasca sigui mes ràpida o més lenta.

4.2 Polítiques a aprendre

Els següents dos punts són els problemes a modelitzar perquè l'Agent aprengui la política adequada per complir-los.

4.2.1 Autonomia en el moviment

L'NPC ha de ser capaç de moure d'esquerra a dreta de manera cíclica. El radi de desplaçament respecte el punt origen ha de ser configurable.

Arribar al màxim rang de desplaçament tant d'esquerra a la dreta ha de donar feedback positiu, però no ha de passar que es quedi en un extrem sense moure, només s'ha de poder atorgar feedback positiu si s'ha vingut de l'altre extrem, un cas excepcional seria la primera vegada que apareix en l'entorn del joc ja que surt del centre del desplaçament.

En cas de col·lisions amb objectes que no siguin targets d'atac dintre del seu rang ha de poder ser capaç de girar i fer el recorregut contrari.

4.2.2 Saber atacar

Un cop l'Agent hagi après a moure's, la meua idea és que l'NPC detecti entitats a atacar.

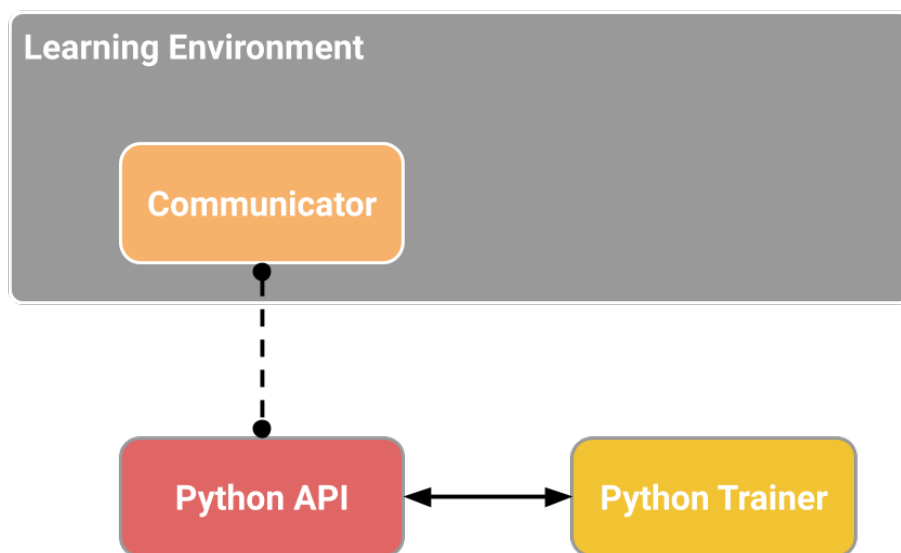
Aquestes entitats únicament seran detectables si estan dintre d'un rang esfèric que tindrà com radi el rang de desplaçament del NPC. L'objectiu en aquest cas és que, deixi de fer el moviment cíclic d'esquerra a dreta i que salti sobre l'objectiu fent col·lisió, les col·lisions encertades donaran feedback positiu.

4.3 Arquitectura, Algorismes, Propostes existents

La arquitectura del problema que abordo be donat per les eines que utilitzo per resoldre-ho, en aquest cas *ML-Agents Toolkit*. A part de les eines de *ML-Agents Toolkit* faré servir *Tensorboard* per veure les estadístiques d'aprenentatge.

A continuació explicaré com es l'arquitectura de ML-Agents Toolkit. ML-Agents Toolkit té els següents quatre blocks com a estructura principal.

- Espai d'aprenentatge (Escena d'Unity)
- Comuniador (Broquer que connecta Unity amb l'API de Python)
- Python API
- Python Trainers



4.3.1 Espai d'aprenentatge

Aquí esta l'escena d'Unity i els personatges. Els personatges que per requeriment del problema s'han definits com Agents, podran observar, prendre decisions i tenir recompenses.

4.3.2 Agent

Objecte d'Unity que hereta de la classe *Agent* amb metodes que poden ser sobrecrits.

La seva API contempla mètodes per poder generar observacions del medi, per prendre accions i per assignar recompenses.

Cada Agent esta relacionat amb un Behavior Parameter class i a un Decision Requester.

Els metodes més comuns a sobrescriure de la classe *Agent* són els següents:

- Initialize
- OnEpisodiBegin
- OnActionReceived
- CollectObservations
- Heuristic

4.3.2.1 Initialize

Aquest mètode es crida un cop únicament en el cicle de vida del aprenentatge. Esta pensat per fer caching d'objectes de l'escena o inicialitzar valors.

4.3.2.2 OnEpisodiBegin

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.

4.3.2.3 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.

Aquest mètode interactua tan amb el mètode *heuristic* i amb el motor d'aprenentatge automàtic.

4.3.2.4 CollectObservations

Aquest mètode s'executa cada x temps, aquest temps d'execució es configura amb l'eina de Unity.

El mètode te com a paràmetre una estructura de dades semblants com a paràmetre d'entrada que el mètode `OnActionReceived`. En aquest cas no esta pensat per consumir les dades sinó per modificar aquesta referencia i per donar informació al motor de Reinforcement Learning quan estem en mode de *Learning*.

4.3.2.5 Heuristic

Aquest mètode permet interactuar amb el jugador. Fixat que te la mateixa estructura de dades com a paràmetre que el mètode `ActionBuffers`.

Els valors que s'enplenin en els buffers d'aquesta estructura de dades aniran a parar al mètode *OnActionReceived*.

4.3.3 Behaviour

Script que esta pensat per parametritzar i relacionar l'Agent amb la configuració externa a Unity.

Un Behavior pot tenir els següents comportaments:

- Learning

- Heuristic
- Inference

4.3.3.1 Learning

Quan el *Behavior* esta en mode *Learning* vol dir que utilitza el model generat per la sobrescritura dels metodes de la classe *Agent* per aprendre. En aquest mode Unity es connecta amb el procés de *mlagents-learn* per generar la xarxa neuronal que representa l'aprenentatge del model.

4.3.3.2 Heuristic

En aquest mode no hi ha procés d'aprenentatge, s'utilitza la funcio *heuristic* de la classe *Agent* per interactuar amb l'Agent.

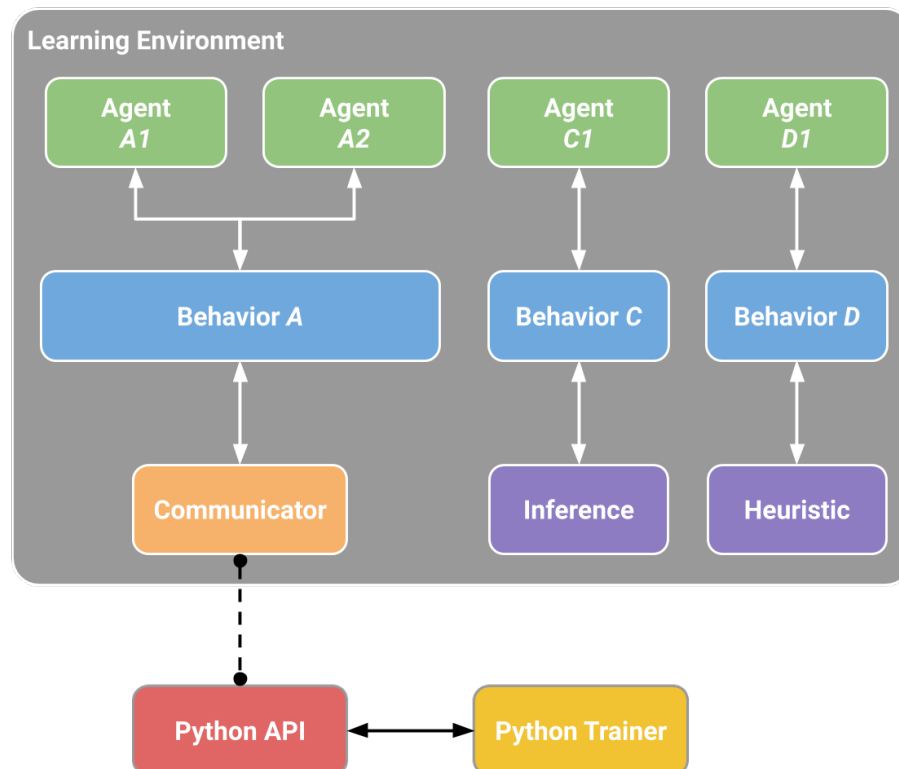
4.3.3.3 Inference

Quan el *Behavior* està en mode *Inference* implica que hi ha una xarxa neuronal amb una política apresada, i l'objecte que conte aquesta xarxa neuronal es comportarà segons lo que s'hagi apres.

4.3.4 Requester

Es un Script que serveix per forçar que l'Agent prengui decisions, sense aquest Script com a Component en el nostre Agent, l'Agent mai prendrà decisions. Aquí es pot configurar el període de decisions.

En la següent imatge hi ha les tres diferents arquitectures que podem tenir segons quin comportament tingui el component *Behavior* del objecte Agent.



4.3.5 Tensorboard

Quan l'entrenament s'està executant, ML-Agents Toolkit guarda estadístiques dintre del directori `results` en un directori amb nom únic, donat a l'hora d'executar el procés `mlagents-learn`.

Per observar el procés d'entrenament quan esta corrent el procés o quan no, ens apropem a la carpeta de resultats i executem: `tensorboard --logdir */results${training id}`.

Això obrirà un servidor per servir el contingut estàtic en un dels ports de la teva màquina.

Explicaré alguna de les gràfiques que *Tensorboard* genera ja que molt segurament les veurem més endavant en cada un de les versions del Agent.

4.3.5.1 Cumulative Reward

Gràfica que representa la mitja de lo que guanya l'Agent per episodi. Un bon procés d'aprenentatge genera una gràfica ascendent que en algun moment s'estanca ja que la seva mitja per episodi es mante.

4.3.5.2 Episodi Length

Representa la durada mitja per episodi. Aquesta gràfica depen de l'implementació del model ja que segons la característica del problema es pot forçar el reinici de episodi, o potser el problema es màximitzar o minimitzar el temps de vida del Agent.

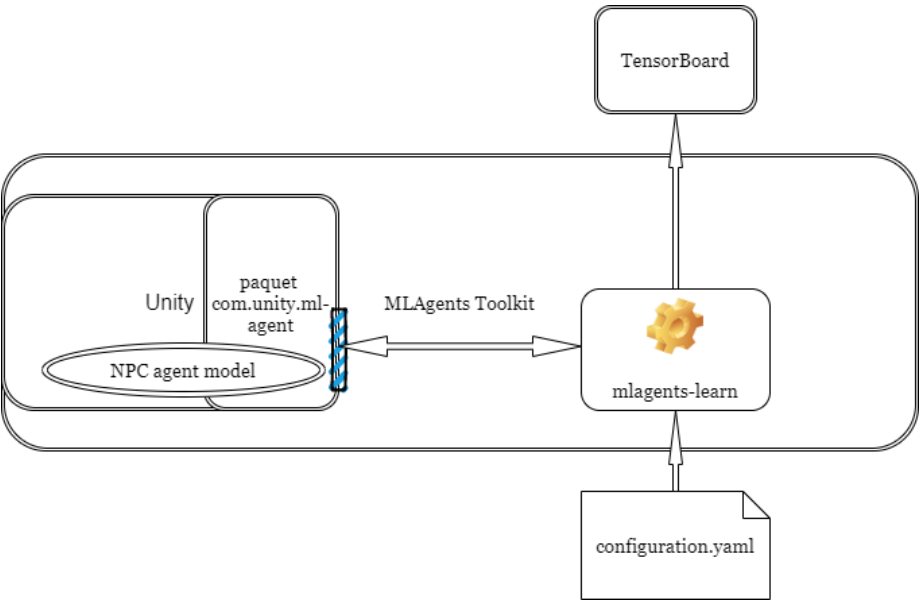
4.3.5.3 Policy Loss

Representa quan la política del Agent canvia, un model correcte tendirà a fer una gràfica descendent.

4.3.5.4 Entropy

Representa que tan aleatoris són els moviments del Agent, un model correcte expresa una gràfica d'Entropia descendent ja que significa que realment esta aprenent una política i no esta fent tants d'intents per provar coses noves.

Arquitectura final



5 Experiments

Aquest projecte conté un seguit d'experiments bàçats en fases. Les fases benen donades pel l'apartat *4.2 Polítiques a aprendre*.

5.1 Representació del objecte NPC dintre d'Unity

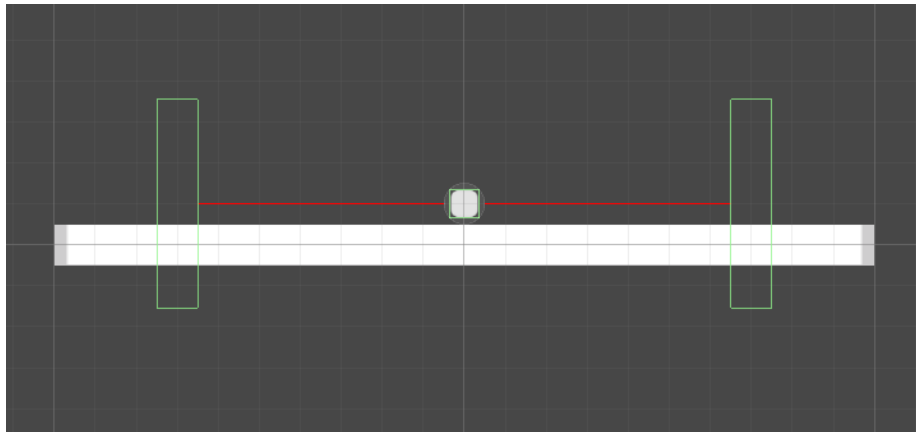
L'objecte NPC esta format tres objectes fills.

- Limit esquerra
- Limit dret
- Agent
 - Raig esquerra
 - Raig dret

Els objectes de limits, son objectes buits i invisibles que tenen un comportament de trigger de events en cas de col·lisions. Aquests objectes son escencials per poder delimitar el rang de moviment del Agent.

L'objecte Agent conte el Script de modelatge que hereta de la classe *Agent* a més dels Scripts *Behavior Parameters* i *DesitionRequester*.

Finalment tenim els rajos, que interaccionen amb elements de l'escana. El funcionament d'aquests rajos s'explica en l'apartat *4.1.1.1 Observacions*.



5.2 Autonomia en el moviment

5.2.1 Versió (I)

Els mètodes que s'han sobrescrit de la classe *Agent* són els següents:

- Initialize
- OnEpisodiBegin
- OnActionReceived
- CollectObservations
- Heuristic

Altres mètodes de classe no propis de la classe *Agent* i que són interessants per poder explicar la modelització del problema:

- OnTriggerEnter2D
- OnTriggerExit2D

5.2.1.1 Initialize

Rutina que s'executa només un cop quan es dona play al motor d'Unity.

Agafa el component del objecte que em permet treballar amb les físiques de Unity. El component es el *RigidBody* que ha de haver estat afegit al objecte com una component desde l'interfície d'Unity. El component *RigidBody* ha de tenir el comportament de *Dynamic* perquè sigui afectat per la gravetat de configurada en l'escena.

Per altra banda configuro el nombre de passos per episodi a zero, en cas que l'estat de la variable *trainingMode* sigui fals. Aquesta configuració em permet fer proves en mode *Heuristic* o en mode *Inferencia* sense passar pel cicle de vida *OnEpisodoBegin*.

La variable *MaxStep* es una variable propia de la classe *Agent* i representa el nombre de passos màxims que es fan quan l'Agent esta aprenent una política. Quan s'ha fet aquests nombres de passos, immediatament pel cicle de vida de l'Agent, força l'execució de la rutina *OnEpisodiBegin*.

```
1 public override void Initialize()  
2 {  
3     _rigidbody2D = GetComponent<Rigidbody2D>();  
4     // infinite steps for session  
5     if (!trainingMode) MaxStep = 0;  
6 }  
7
```

5.2.1.2 OnEpisodeBegin

El següent mètode s'executa al inici de cada episodi. És important de restablir els estats inicials d'algunes propietats de l'Agent, degüt a que l'Agent manté la inèrcia del episodi anterior.

En la rutina es treu l'inèrcia que tenia l'Agent en el episodi anterior, s'el reposiciona al centre i s'escull una meta aleatoriament. A aquestes altures les metes a escollir són els límits.

```
1 public override void OnEpisodeBegin()
2 {
3     // resetting movement inercy
4     _rigidbody2D.velocity = Vector2.zero;
5     //reseting positions
6     transform.position = transform.parent.position;
7     // changin randomnes
8     Random.InitState(System.DateTime.Now.Millisecond);
9     // finding the moving target
10    FindMovingTarget();
11 }
12
```

5.2.1.3 OnActionReceived

La lògica dintre d'aquests mètode determina el comportament de l'Agent. Les accions que s'esperen es una continua, un valor entre menys u i més u. Segons aquesta acció la velocitat del Agent serà donada.

```
1 // called when action is received from either {player,
2 // neural network}
3 // each buffer position refers to an action, I decide what
4 // it means for each positions
5 // inside that structure has continuous and discrete
6 // actions
7 // index 0: -1 means move to the left, +1 means move to
8 // the right
9 // the cool thing about the neural network, is that it
10 // figures it all automatic
11 public override void OnActionReceived(ActionBuffers
12 actions)
13 {
14     Vector2 movement = new Vector2(actions.ContinuousActions
15 [0] * movementForce, 0);
16     _rigidbody2D.velocity = movement;
17 }
18
```

5.2.1.4 CollectObservations

En aquest mètode li he de donar al procés d'aprenentatge aquells paràmetres que jo crec que són importants per tenir en compte a l'hora d'aprendre la política.

Aquí vaig trobar interessant d'informar al procés d'aprenentatge, quina es la horientació del Agent respecte el punt objectiu. La distancia del Agent respecte el punt objectiu, la posició del punt objectiu i finalment la velocitat de moviment del Agent.

```
1 // Should include all variables relevant for following
2 // to take the agent the optimally informed desition.
3 // No extraneous information here please
4 public override void CollectObservations(VectorSensor
  sensor)
5 {
6     if (!currentTarget) return;
7
8     Vector2 currentPos = new Vector2(transform.position.x,
9     0);
10    Vector2 targetPos = new Vector2(currentTarget.position.x
11    , 0);
12
13    Vector2 toTarget = targetPos - currentPos;
14    // 2 observations (horientation)
15    sensor.AddObservation(toTarget.normalized);
16    // 1 observation (distance)
17    sensor.AddObservation(Vector2.Distance(targetPos,
18    currentPos));
19    // 2 observations for current target position
20    sensor.AddObservation(targetPos);
21    // 2 observations for movement velocity
22    sensor.AddObservation(_rigidbody2D.velocity);
23    // Note: curiosamente si normalizo la velocidad, le
24    cuesta mucho aprenderx
25 }
```


5.2.1.5 Heuristic

En aquesta versió únicament l'Agent ha de poder interpretar valors entrats per teclat que representin moviments laterals.

Es modifica la referència dels *ActionBuffers* de tal manera que en el buffer de variables continues es passa el valor entrat per teclat.

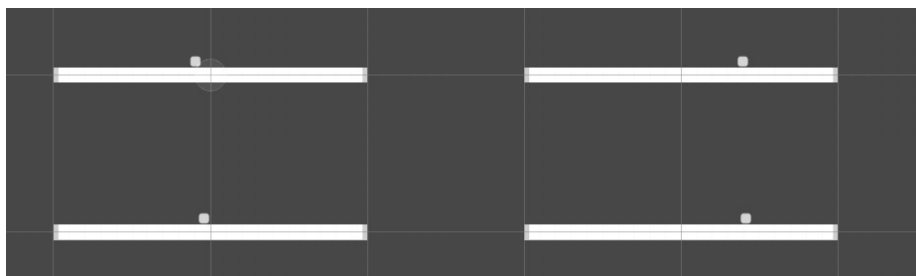
```
1 // this method allows me to interact with the game
2 // when the ml agents is not set to training
3 public override void Heuristic(in ActionBuffers actionsOut
4 )
5 {
6     ActionSegment<float> continuosActions = actionsOut.
7     ContinuousActions;
8     var force = Input.GetAxis("Horizontal");
9     continuosActions[0] = force;
10 }
```

5.3 Metodologia i Resultats

5.3.1 Autonomia en el moviment, versió (I)

L'objectiu de fer aprendre la política de moures ciclacament ha sigut tot un èxit.

Perque el temps d'aprenentatge es fes més curt, el que vaig fer va ser duplicar l'Agent quatre vegades.



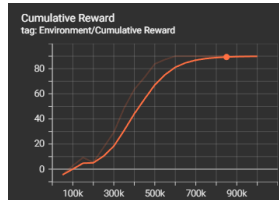
Fixem-nos en els següents logs que *mlagents-learn* va soltant en el procés d'aprenentatge.

```
[INFO] Resuming from results\ppo\NPC.
[INFO] Resuming training from step 896.
[INFO] NPC. Step: 50000. Time Elapsed: 149.945 s. Mean Reward: -4.500. Std of Reward: 13.955. Training.
[INFO] NPC. Step: 100000. Time Elapsed: 230.711 s. Mean Reward: 2.791. Std of Reward: 8.714. Training.
[INFO] NPC. Step: 150000. Time Elapsed: 303.819 s. Mean Reward: 9.286. Std of Reward: 5.517. Training.
[INFO] NPC. Step: 200000. Time Elapsed: 368.859 s. Mean Reward: 5.349. Std of Reward: 12.454. Training.
[INFO] NPC. Step: 250000. Time Elapsed: 447.106 s. Mean Reward: 17.317. Std of Reward: 13.619. Training.
[INFO] NPC. Step: 300000. Time Elapsed: 524.144 s. Mean Reward: 29.286. Std of Reward: 18.695. Training.
[INFO] NPC. Step: 350000. Time Elapsed: 593.135 s. Mean Reward: 48.537. Std of Reward: 13.715. Training.
[INFO] NPC. Step: 400000. Time Elapsed: 673.271 s. Mean Reward: 63.488. Std of Reward: 11.790. Training.
[INFO] NPC. Step: 450000. Time Elapsed: 754.590 s. Mean Reward: 73.171. Std of Reward: 9.223. Training.
[INFO] NPC. Step: 500000. Time Elapsed: 834.312 s. Mean Reward: 83.810. Std of Reward: 6.884. Training.
[INFO] Exported results\ppo\NPC\NPC-499932.onnx
[INFO] NPC. Step: 550000. Time Elapsed: 914.108 s. Mean Reward: 87.317. Std of Reward: 4.431. Training.
[INFO] NPC. Step: 600000. Time Elapsed: 993.981 s. Mean Reward: 90.000. Std of Reward: 0.000. Training.
[INFO] NPC. Step: 650000. Time Elapsed: 1074.479 s. Mean Reward: 90.000. Std of Reward: 0.000. Training.
[INFO] NPC. Step: 700000. Time Elapsed: 1152.584 s. Mean Reward: 90.000. Std of Reward: 0.000. Training.
[INFO] NPC. Step: 750000. Time Elapsed: 1233.012 s. Mean Reward: 90.000. Std of Reward: 0.000. Training.
[INFO] NPC. Step: 800000. Time Elapsed: 1312.612 s. Mean Reward: 90.000. Std of Reward: 0.000. Training.
[INFO] NPC. Step: 850000. Time Elapsed: 1394.562 s. Mean Reward: 90.000. Std of Reward: 0.000. Training.
[INFO] NPC. Step: 900000. Time Elapsed: 1546.801 s. Mean Reward: 90.000. Std of Reward: 0.000. Training.
[INFO] NPC. Step: 950000. Time Elapsed: 1628.662 s. Mean Reward: 90.000. Std of Reward: 0.000. Training.
[INFO] NPC. Step: 1000000. Time Elapsed: 1706.933 s. Mean Reward: 90.000. Std of Reward: 0.000. Training.
```

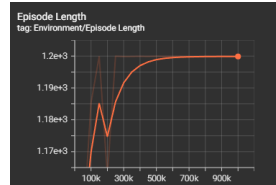
Cada una de les entrades del log representen episodis. Les primeres entrades donen una mitja *Mean Reward* que les últimes i la desviació estàndard es bastant més ample que les últimes.

Finalment arriba el moment on l'aprenentatge de l'Agent no canvia. Això vol dir que ha après una política i immediatament, *mlagents-learn* quan veu que la mitja per cada episodi es la mitja tanca el procés d'aprenentatge.

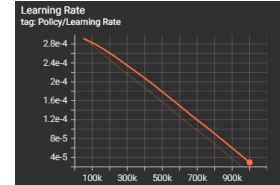
A continuació veurem algunes gràfiques que *Tensorboard* ens ha generat respecte aquest procés d'aprenentatge.



(a) Cumulative reward



(b) Episodi length



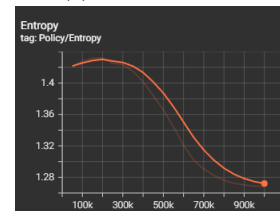
(c) Learning rate



(d) Policity loss



(e) Value Loss



(f) Entropy

6 Conclusions i línies futures

7 Referències