

# **Aprendentatge per reforç amb Unity**

## TAIA Project

Wilber Eduardo Bermeo Quito

12 de desembre de 2021

# Índex

<b>1 Problema a resoldre en una frase</b>	<b>3</b>
<b>2 Introducció i Motivació</b>	<b>3</b>
<b>3 Dades/Coneixements dels que es disposa</b>	<b>3</b>
<b>4 Proposta</b>	<b>4</b>
4.1 Agent . . . . .	4
4.1.1 Com 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 . . . . .	9
4.3.4 Requester . . . . .	10
4.3.5 Tensorboard . . . . .	11
<b>5 Experiments</b>	<b>13</b>
5.1 Representació de l'objecte NPC dintre d'Unity . . . . .	13
5.1.1 Versió 1 . . . . .	13
5.1.2 Versións 2.1, 2.2, 2.3 . . . . .	14
5.2 Metodologia i Resultats . . . . .	15
5.2.1 Versió 1 . . . . .	15
5.2.2 Versió 2.1 . . . . .	20
5.2.3 Versió 2.2 . . . . .	24
5.2.4 Versió 2.3 . . . . .	27
<b>6 Conclusions i línies futures</b>	<b>31</b>
<b>7 Referències</b>	<b>32</b>

## 1 Problema a resoldre en una frase

Fer que un NPC segueixi una política en concret fent ús 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 és 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

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

Conec l'entorn de treball Unity, no professionalment, però si tinc un base.

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

## 4 Proposta

### 4.1 Agent

Classe que conté 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 està relacionat amb un Behavior Parameter class i a un Decision Requester.

#### 4.1.1 Com 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 o visuals. Les observacions numèriques són del punt de vista de l'observador (NPC), les visuals en canvi, són generades per col·lisions de rajos de sensors que es poden ajuntar a l'agent per que tingui informació del terreny i altres objectes en la escena, representen el que l'agent està veient en aquell moment.

##### 4.1.1.2 Accions

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

##### 4.1.1.3 Recompenses

És un escalar que representa que tan bé ho està fent l'agent. Les recompenses (negatives o positives) no tenen per què ser donades a cada moment, es poden donar en certs moments provocats per esdeveniments. És important saber quan i quant es recompensa o es castiga a l'agent pel seu acte. Depenen de la política de recompenses farà que l'aprenentatge d'una certa tasca sigui més 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

(v1)

L'NPC ha de ser capaç de moure d'esquerra a dreta de manera cíclica.

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 i 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. (no es fa).

(v2.1, v2.2, v2.3)

L'Agent a més a més de desplaçar-se d'esquerra a dreta ara té l'habilitat de poder fer salts. Els salts únicament estan pensats per canviar a mode d'atac. En cas de fer un salt innecessari, l'Agent és veurà castigat. D'aquest amanerà mantindrà el moviment cíclic de la primera versió, però tindrà la capacitat de saltar. Que es considera com a salt innecessari?, doncs saltar quan no està sobre el terra o quan no hi hagi enemics detectats.

### 4.2.2 Saber atacar

(v2.1, v2.2, v2.3)

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

Aquestes entitats únicament seran detectades pels rajos del sensor que el NPC té incorporat. 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.

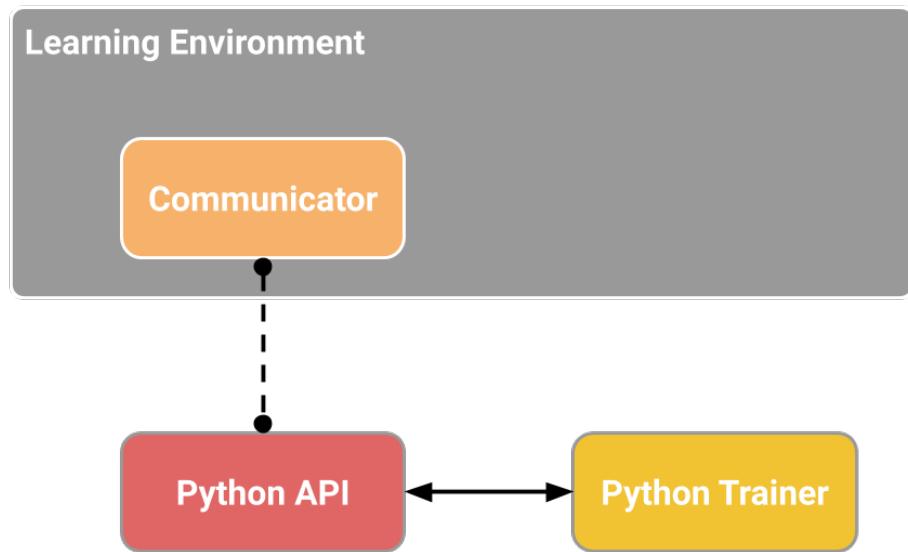
L'agent mor si és tocat per l'enemic quan aquest no està en mode atac que és el mateix a dir, no haver fet el salt moments abans de la col·lisió amb l'enemic.

### 4.3 Arquitectura, Algorismes, Propostes existents

L'arquitectura del problema que abordo bé 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 és l'arquitectura de ML-Agents Toolkit. ML-Agents Toolkit té els següents quatre blocs 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 està l'escena d'Unity i els personatges. Els personatges que per requeriment del problema s'han definit com a Agents, podran observar, prendre decisions i tenir recompenses.

#### **4.3.2 Agent**

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

- Initialize
- OnEpisodiBegin
- OnActionReceived
- CollectObservations
- Heuristic

#### **4.3.2.1 Initialize**

Aquest mètode es crida un cop únicament en el cicle de vida de l'aprenentatge. Està 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 de l'episodi s'han completat.

#### **4.3.2.3 OnActionReceived**

Aquest mètode porta un paràmetre d'entrada, anomenat ActionBuffer, l'ActionBuffer, és un objecte que té dos arrays, un per poder representar valors continus i altres per representar valors discrets.

Aquest mètode interactua tant 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 d'Unity.

El mètode té com a paràmetre una estructura de dades semblants com a paràmetre d'entrada que el mètode OnActionReceived. En aquest cas no està pensat per consumir les dades sinó per modificar aquesta referència 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 té la mateixa estructura de dades com a paràmetre que el mètode ActionBuffers.

Els valors que s'emplenin 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* està en mode *Learning* vol dir que utilitza el model generat per la sobreescritura dels mètodes 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, es fa servir la funció *heuristic* de la classe *Agent* per interactuar amb l'Agent.

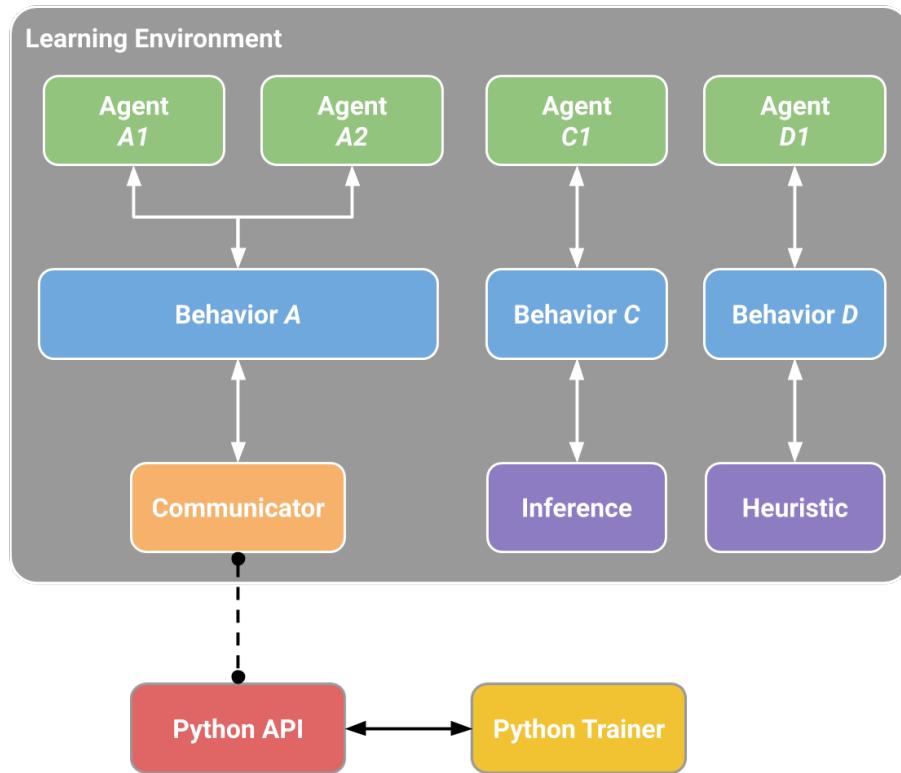
#### **4.3.3.3 Inference**

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

#### 4.3.4 Requester

És 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 resolts en un directori amb nom únic, donat a l'hora d'executar el procés `mlagents-learn`.

Per observar el procés d'entrenament quan està 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 de l'Agent.

#### 4.3.5.1 Cumulative Reward

Gràfica que representa la mitja de el que guanya l'Agent per episodi. Un bon procés d'aprenentatge gènera una gràfica ascendent que en algun moment s'estanca, ja que la seva mitja per episodi és manté.

#### 4.3.5.2 Learning Rate

Representa el temps que li pren per pas a l'algoritme d'entrenament per trobar la política òptima. Hi ha de decréixer linealment en cas que l'aprenentatge s'estigui fent correctament.

#### 4.3.5.3 Episodi Length

Representa la durada mitjana per episodi. Aquesta gràfica depèn de la implementació del model, ja que segons la característica del problema es pot forçar el reinici d'episodi, o potser el problema és maximitzar o minimitzar el temps de vida de l'Agent.

#### 4.3.5.4 Policy Loss

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

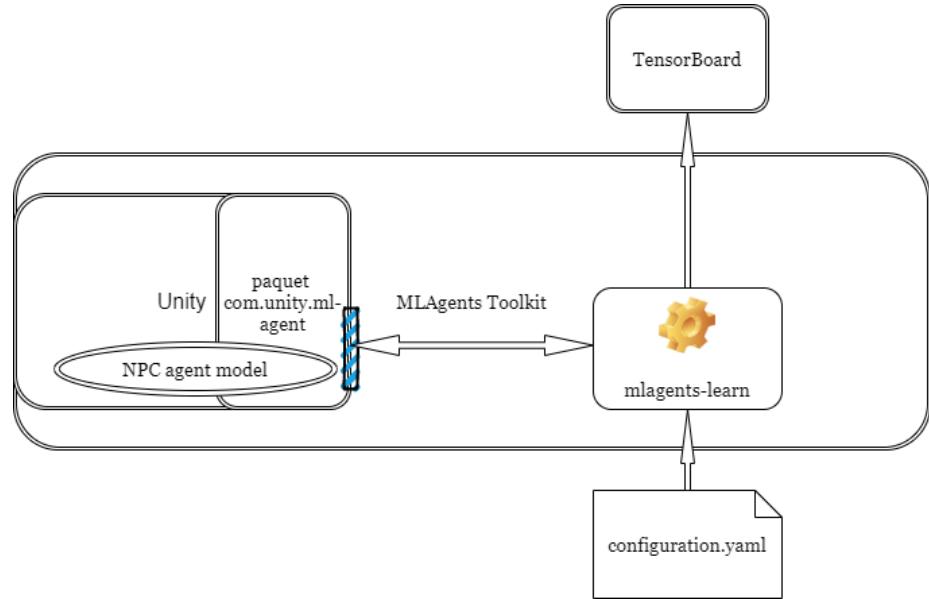
#### 4.3.5.5 Value Loss

Representa que tan bé el model es capaç de predir el valor de cada estat. La gràfica ha d'incrementar mentre l'Agent aprèn, però quan la mitja de premi s'estableix ha de descendir

#### 4.3.5.6 Entropy

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

**Arquitectura final**



## 5 Experiments

Aquest projecte conté un seguit d'experiments basats en fases. Les fases venen donades per l'apartat *4.2 Polítiques a aprendre*.

### 5.1 Representació de l'objecte NPC dintre d'Unity

L'objecte NPC està format tres objectes fills.

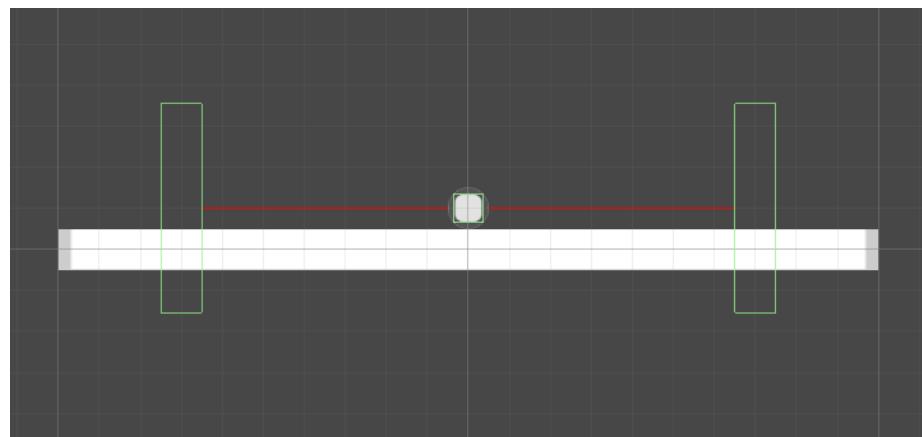
#### 5.1.1 Versió 1

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

Els objectes de límits, són objectes buits i invisibles que tenen un comportament de trigger de esdeveniments en cas de col·lisions. Aquests objectes són essencials per poder delimitar el rang de moviment de l'Agent.

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

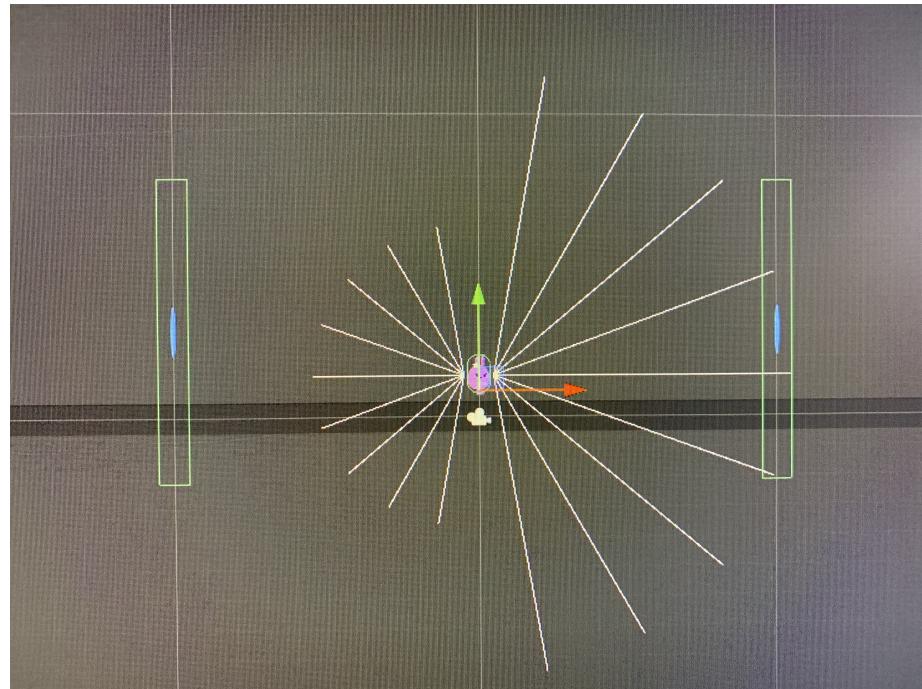
Finalment tenim els rajos, que interaccionen amb elements de l'escena. El funcionament d'aquests rajos s'explica en l'apartat *4.1.1.1 Observacions*.



### 5.1.2 Versións 2.1, 2.2, 2.3

L'arquitectura en Unity del NPC és manté, però canvien la quantitat de rajos de raytraicing llençats pels dos sensors. Noteu en la següent imatge que:

- Els rajos frontals són més llargs que els posteriors, d'aquesta manera es dona més realisme al fet que es veu més cap endavant que no a les espalles
- Els rajos estan dispersats en forma de ventall per cobrir més terreny i saber si algu a entrat a la zona d'atac.



## 5.2 Metodologia i Resultats

### 5.2.1 Versió 1

Els mètodes que s'hi han sobreescrit 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.

Agafo el component de l'objecte que em permet treballar amb les físiques de Unity. El component és el *RigidBody* que ha d'haver estat afegit a l'objecte com una component des de 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 *OnEpisodiBegin*.

La variable *MaxStep* és una variable pròpia de la classe *Agent* i representa el nombre de passos màxims que es fan durant l'Agent està 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 (!trainningMode) MaxStep = 0;
6  }
7
```

### 5.2.1.2 OnEpisodeBegin

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

En la rutina es treu l'inèrcia que tenia l'Agent en l'episodi anterior, se'l 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 és una continua, un valor entre menys u i més u. Segons aquesta acció la velocitat de l'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 }
```

#### 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 el procés d'aprenentatge, quina és la horientació del Agent respecte el punt objectiu. La distància de l'Agent respecte el punt objectiu, la posició del punt objectiu i finalment la velocitat de moviment de l'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
5     sensor)
6 {
7     if (!currentTarget) return;
8
9     Vector2 currentPos = new Vector2(transform.position.x,
10    0);
11    Vector2 targetPos = new Vector2(currentTarget.position.x
12    , 0);
13
14    Vector2 toTarget = targetPos - currentPos;
15    // 2 observations (horientation)
16    sensor.AddObservation(toTarget.normalized);
17    // 1 observation (distance)
18    sensor.AddObservation(Vector2.Distance(targetPos,
19    currentPos));
20    // 2 observations for current target position
21    sensor.AddObservation(targetPos);
22    // 2 observations for movement velocity
23    sensor.AddObservation(_rigidbody2D.velocity);
24    // Note: curiosamente si normalizo la velocidad, le
25    // cuesta mucho aprenderx
26}
```

### 5.2.1.5 Heuristic

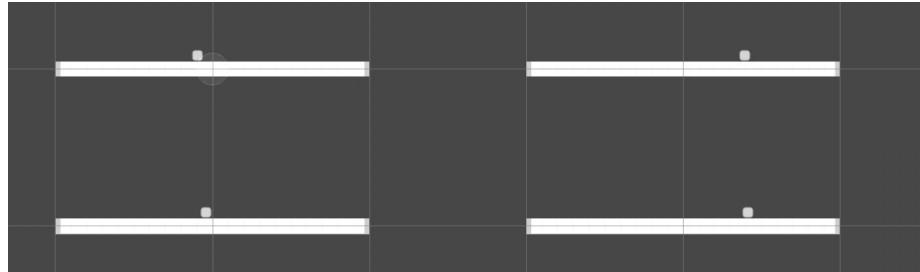
En aquesta versió únicament l'Agent ha de poder interpretar valors entrats per teclat que representin moviments laterals.

Es modifica la referència dels *ActionBuffers* de tal manera que en el buffer de variables contínues és 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;
}
```

L'objectiu de fer aprendre la política de moure's cíclicament ha sigut tot un èxit.

Perquè 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.

```

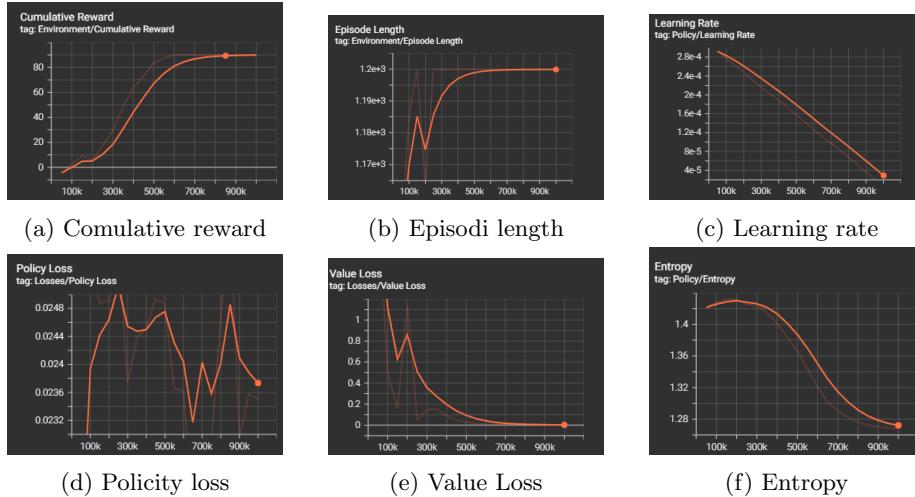
[INFO] Resuming from results\ppo\NPC.
[INFO] Resuming training from step 896.
[INFO] NPC. Step: 50000. Time Elapsed: 149.945 s. Mean Reward: -4.500. Std of Reward: 13.955. Training.
[INFO] NPC. Step: 100000. Time Elapsed: 230.711 s. Mean Reward: 2.791. Std of Reward: 8.714. Training.
[INFO] NPC. Step: 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: 11.798. 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 és 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 és la mateixa tanca el procés d'aprenentatge.

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



### 5.2.2 Versió 2.1

La versió 1 i la versió 2.1 i superiors no són compatibles perquè s'ha incrementat el nombre de rajos per sensor de detecció, i Unity es queixa i no deixa reutilitzar la xarxa neuronal de la versió 1.

L'Agent en la versió 2.1 és capaç de fer salts i ja va estar preparat per poder atacar a possibles enemics (encara que aquestes funcionalitat no està polida). A la versió 2.1 no se li incorpora interactivitat amb enemics, es fan les proves per veure si amb els canvis que s'han fet respecta versió 1, el NPC segueix sent capaç de fer el desplaçament cíclic.

Els canvis que s'han fet en la versió 2.1 fan que el NPC tingui una interacció de tipus continua (0,1) que representa el fet de saltar. Si s'activa l'objecte HeroGenerator de l'escena podem veure que l'Agent tindrà dificultats, ja que els objectes generats col·lisionaran amb ell.

Aquesta versió sobreescrivia els mateixos mètodes de la classe Agent, però amb refactoritzacions i ampliació de lògica en alguna d'elles.

Mètodes sobrescrits que és manté igual:

- Collect observations
- Initialize

#### 5.2.2.1 OnEpisodeBegin

En aquesta versió es pot veure quan l'Agent a col·lisionat amb un dels límits, cada cop que s'inicialitza un entrenament, aquest els torna al seu estat inicial. Per la resta és pràcticament igual excepte per la crida a la funció PickOneLimitAsTarget que abans es deia FindMovingTarget.

```
1  public override void OnEpisodeBegin()
2  {
3      // resetting movement inercy
4      rb.velocity = Vector2.zero;
5      //reseting positions
6      transform.position = transform.parent.position;
7      // changin randomnes
8      Random.InitState(System.DateTime.Now.Millisecond);
9      // current target rebooted
10     target = null;
11     // setting attack mode to false
12     attackMode = false;
13     // finding the moving target
14     PickOneLimitAsTarget();
15     // restart limits
16     RestartLimits();
17 }
18
```

### 5.2.2.2 OnActionReceived

Aquest és un dels mètodes que ha canviat més. Obliga al fet que l'Agent faci la tasca el ràpid possible, castiga a l'Agent en cas de fer salts innecessaris i premia en cas que l'orientació donada per l'aprenentatge estigui orientada cap al target.

```
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 continuos 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 // figurates it all automatic
11
12 // the second element is discrete parameter, which says
13 // jump or not [0,1]
14 // when npc jumps has the hability to make damage and only
15 // should jump when there is a enemy nearby
16 // and it is in the ground.
17
18 // in case it is not in the ground and there is no enemy
19 // nearby I penalize the agent
20
21 // as I want to make the agent move to target as quick as
22 // possible, each time
23 // on action received is executed I penalize him
24
25 // in case the action given to move target is well
26 // oriented I give him a little reward
27 public override void OnActionReceived(ActionBuffers
28     actions)
29 {
30     int jump = actions.DiscreteActions[0];
31     float xorientation = actions.ContinuousActions[0];
32     Vector2 v = rb.velocity;
33
34     AddReward(-gain / 10000); // trying to finish the task
35     quickly
36
37     // checking if the new orientation action is good
38     if (target)
39     {
40         Vector2 orientation = new Vector2(xorientation, 0).
41         normalized;
42         float simil = Vector2.Dot(orientation,
43             TargetDirectionNormalized());
44
45         if (simil > Mathf.Epsilon)
46         {
47             Debug.Log("Well oriented");
48             Debug.Log(simil);
49         }
50     }
51 }
```

```

35     AddReward(gain / 100);
36 }
37 }
38 }
39 // try to learn not to jump when it is not in the ground
40 bool unneededJump = jump == 1 && !ShouldJump();
41 if (unneededJump)
42 {
43     Debug.Log("Unneeded jump");
44 }
45 if (trainning && unneededJump)
46 {
47     AddReward(-gain);
48     EndEpisode();
49 }
50
51 // run attack mode
52 if (jump == 1 && !attackMode)
53 {
54     StartCoroutine(AttackModeCoroutine());
55 }
56
57 Vector2 movement = new Vector2(xhorientation *
58 movementPower, jump == 1 ? jump * jumpPower : v.y);
59 rb.velocity = movement;
60 }
61

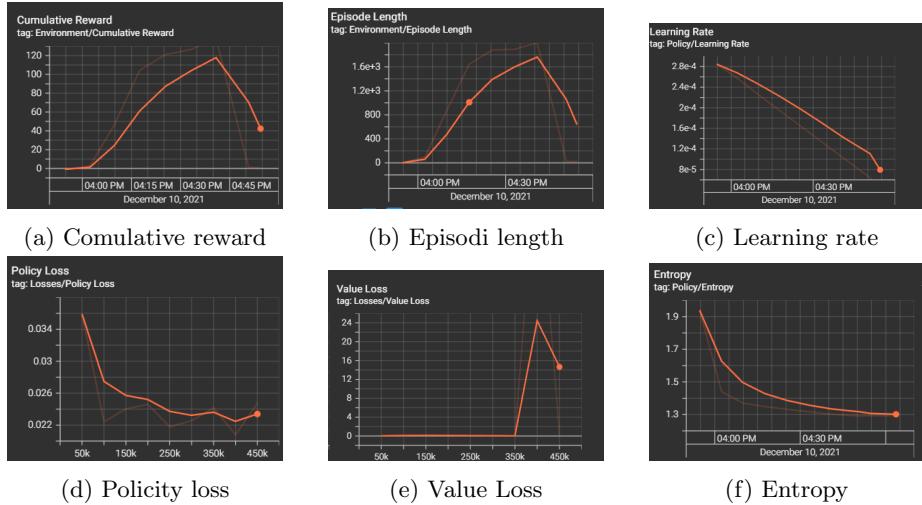
```

### 5.2.2.3 Heuristic

El canvi en aquest mètode hi ha sigut mínim, el canvi principal és permetre al jugador fer un salt amb la tecla Scape si l'Agent està situat a terra.

```
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     ActionSegment<int> discreteActions = actionsOut.
9     DiscreteActions;
10
11     discreteActions[0] = Input.GetKey(KeyCode.Space) &&
12     IsGrounded() ? 1 : 0;
13     continuosActions[0] = Input.GetAxis("Horizontal");
14 }
```

Estadístiques llençades per Tensorboard de l'entrenament de la versió 2.1.



### 5.2.3 Versió 2.2

La versió 2.2 és l'ampliació de la versió 2.1, en aquest cas els processos d'entrenament tindran en compte el factor enemic i es fa que aprengui a no col·lisionar amb ells sense estar en mode atac. Té alguns falles que es milloren en la versió v2.3.

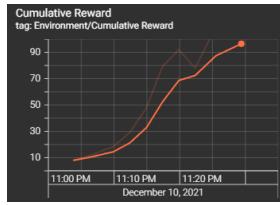
El canvi més significatiu es fa en el mètode OnActionReceived, en la versió 2.1 a l'Agent se'l castigava per saltar quan no havia de fer-ho i s'acabava i es迫ava l'acabament de l'episodi d'aprenentatge, en aquesta versió se'l castiga però de forma més lleu i no s'acava el procés d'aprenentatge, a més a més de donar-li un petit incentiu quan ha fet un salt correcte.

```
1 public override void OnEpisodeBegin()
2 {
3     Debug.Log("New episode...");  
4     // resetting movement inercy  
5     rb.velocity = Vector2.zero;  
6     //reseting positions  
7     transform.position = transform.parent.position;  
8     // changin randomnes  
9     Random.InitState(System.DateTime.Now.Millisecond);  
10    // current target rebooted  
11    // setting attack mode to false`  
12    // removes enimis from scene  
13    if (heroGenerator)  
14    {  
15        heroGenerator.Clean();  
16    }  
17    attackMode = false;  
18    target = null;  
19    // finding the moving target  
20    PickOneLimitAsTarget();  
21    // restart limits  
22    RestartLimits();  
23}  
24  
  
1 // in case it is not in the ground and there is no enemy  
2 // nearby I penalize the agent  
3 // as I want to make the agent move to target as quick as  
4 // possible, each time  
5 // on action received is executed I penalize him  
6 // in case the action given to move target is well  
7 // oriented I give him a little reward  
8 public override void OnActionReceived(ActionBuffers  
9 actions)  
10 {  
11     int jump = actions.DiscreteActions[0];  
12     float xhorientation = actions.ContinuousActions[0];  
13     Vector2 v = rb.velocity;  
14     AddReward(-gain / 10000); // trying to finish the task  
15     quickly
```

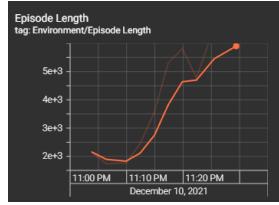
```

11 // checking if the new orientation action is good
12 if (target)
13 {
14     Vector2 orientation = new Vector2(xorientation, 0).
15     normalized;
16     float simil = Vector2.Dot(orientation,
17     TargetDirectionNormalized());
18
19     if (simil > Mathf.Epsilon)
20     {
21         AddReward(gain / 100);
22     }
23 }
24 bool shouldJump = ShouldJump(); // SHOULD JUMP WHEN NOT
25 // try to learn not to jump when it is not in the ground
26 bool unneededJump = jump == 1 && !shouldJump;
27
28 bool goodJump = jump == 1 && shouldJump;
29
30 if (trainning && unneededJump)
31 {
32     Debug.Log("Unneeded jump...");
33     AddReward(-gain / 100);
34     // EndEpisode(); // CLAVE!!!
35 }
36
37 if (trainning && goodJump) // CLAVE
38 {
39     Debug.Log("Good jump...");
40     AddReward(gain / 10);
41 }
42 Vector2 movement = new Vector2(xorientation *
43     movementPower, jump == 1 && shouldJump ? jump * jumpPower
44     : v.y); // CLAVE
45 rb.velocity = movement;
46 // run attack mode
47 if (jump == 1 && !attackMode)
48 {
49     StartCoroutine(AttackModeCoroutine());
50 }
51
52 }
```

Estadístiques llençades per Tensorboard de l'entrenament de la versió 2.2.



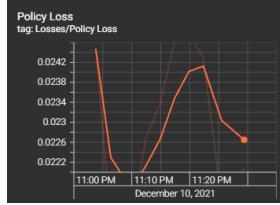
(a) Cumulative reward



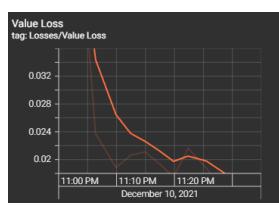
(b) Episodi length



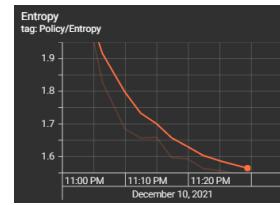
(c) Learning rate



(d) Policy loss



(e) Value Loss



(f) Entropy

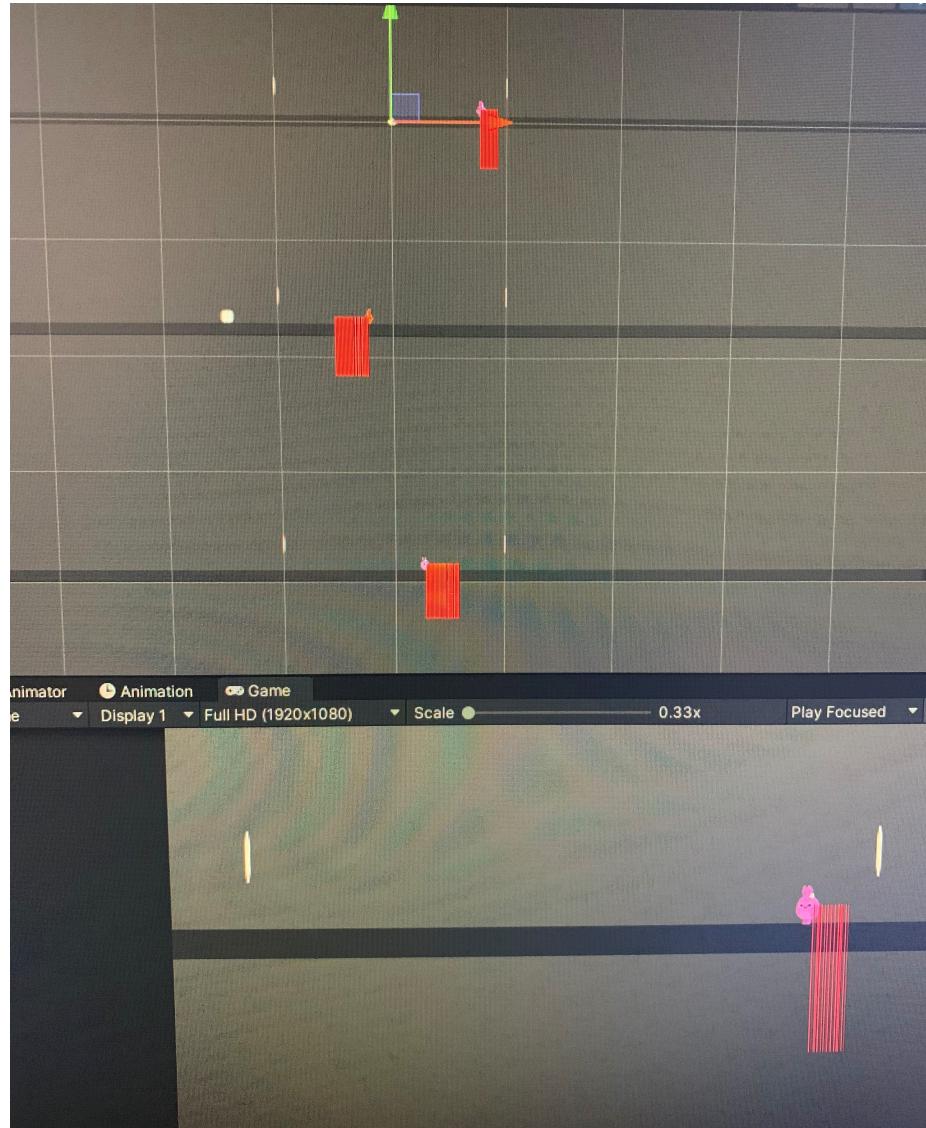
#### **5.2.4 Versió 2.3**

La versió 2.3 té dos processos d'aprenentatge, un amb el fitxer per defecte de configuració de `mlagent` i un fitxer de configuració costum.

La versió 2.3 no és compatible amb les seves menors perquè el vector d'observacions de la xarxa neuronal generada és més gran, ja que percep una variable més, que l'estat atac, que no el de les xarxes de la versions 2.1 i 2.2 no el feien.

La versió 2.3 arregla errors de la versió 2.2 com ara, evitar salts no necessaris, per autodefensa comença a fer múltiples salts, arregla el tema estètic referent al mode atac que es feia per un mal condicional, i finalment el important és la versió 2.3 que serà l'última, té com un dels camps a observar la variable global `attackmode`. D'aquesta manera se li dona més informació a l'algoritme d'aprenentatge, i és rellevant perquè és una variable, que segons el seu estat, es pot o no saltar, es pot o no entrar en mode atac.

En l'última versió, vaig modificar el fitxer de configuració `.yaml` per defecte per multiplicar per deu la sessió d'entrenament, ja que veia que amb els 500000 passos per defecte que feia com a màxim no eren necessaris per aprendre correctament la política, també vaig augmentar el nombre de layers per neurona de dos a tres, i finalment vaig augmentar els `hidden_units` de 128 a 256 per cada layer, a veure si d'aquesta manera el rendiment a trobar els patrons millorava. També vaig haver de fer sessions de treure el generador d'enemics (`herois`) i passar-los, perquè eviti aprendre que saltar per defensar-se sense cap enemic és cosa bona. I ja que puc generar rèpliques de l'escenari, també vaig fer modificacions a cada rèplica com ara, el període generació enemic, en què algunes hi hagi o no enemics i la variació de la velocitat pròpia del NPC.



En aquesta versió el canvi més important és que li dono informació al procés d'aprenentatge de l'estat de l'atac mode i el control correcte d'animacions de salts.

```

1     public override void OnActionReceived(ActionBuffers
2         actions)
3     {
4         int jump = actions.DiscreteActions[0];
5         float xorientation = actions.ContinuousActions[0];
6         Vector2 v = rb.velocity;
7         AddReward(-gain / 10000); // trying to finish the task
8         quickly
9
10        // checking if the new orientation action is good
11        if (target && training)
12        {
13            Vector2 orientation = new Vector2(xorientation, 0).
14            normalized;
15            float simil = Vector2.Dot(orientation,
16            TargetDirectionNormalized());
17            @< -151,7 +151,7 @> public override void OnActionReceived(
18            ActionBuffers actions)
19
20        }
21
22        bool shouldJump = ShouldJump() && !attackMode; // SHOULD
23        JUMP WHEN NOT IN MODE ATTACK TOO
24
25        // try to learn not to jump when it is not in the ground
26        bool unneededJump = jump == 1 && !shouldJump;
27        @< -171,11 +171,10 @> public override void
28        OnActionReceived(ActionBuffers actions)
29        {
30            AddReward(gain / 10);
31        }
32
33        Vector2 movement = new Vector2(xorientation *
34        movementPower, jump == 1 && shouldJump ? jump * jumpPower
35        : v.y); // CLAVE
36        rb.velocity = movement;
37        // run attack mode
38        if (jump == 1 && shouldJump)
39        {
40            StartCoroutine(AttackModeCoroutine());
41        }
42    }
43
44
45    public override void CollectObservations(VectorSensor
46        sensor)
47    {
48        if (!target) return;
49        Vector2 currentPos = new Vector2(transform.position.x,
50        0);
51        Vector2 targetPos = new Vector2(target.transform.

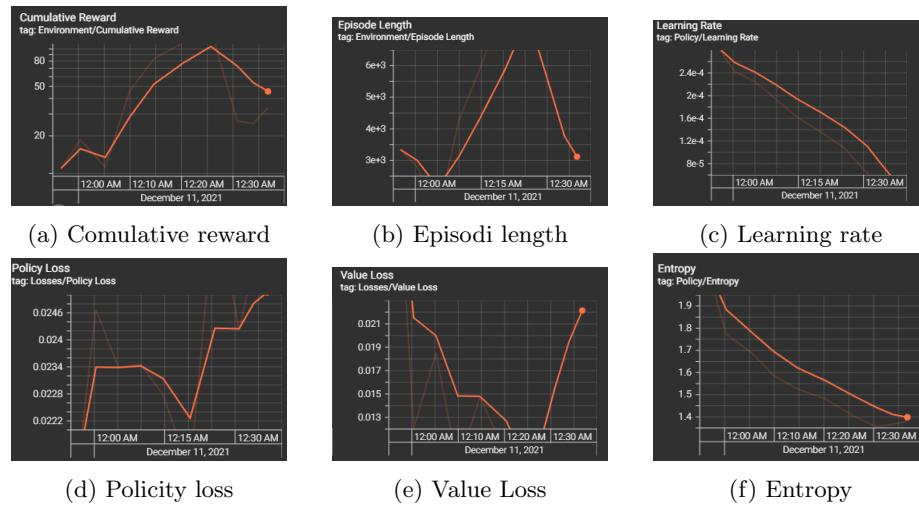
```

```

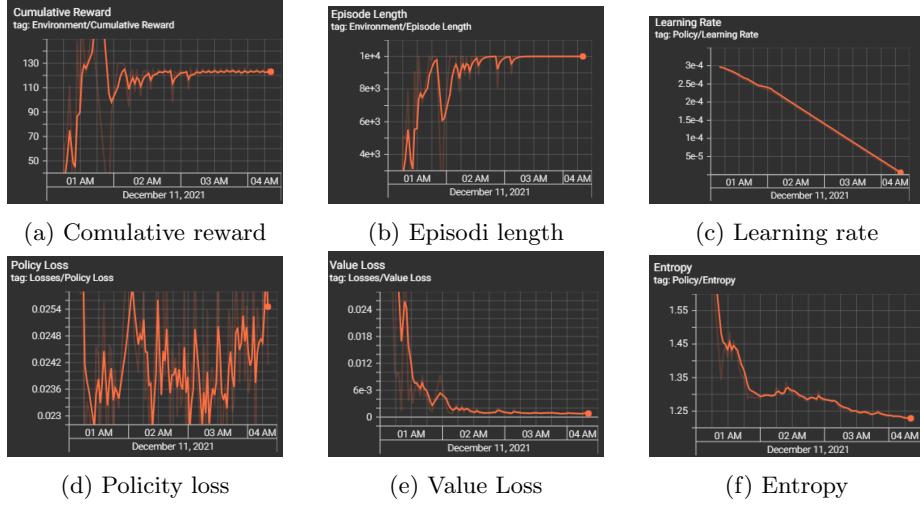
position.x, 0);
Vector2 toTarget = targetPos - currentPos;
// 2 observations (orientation)
sensor.AddObservation(toTarget.normalized);
// 1 observation (distance)
sensor.AddObservation(Vector2.Distance(targetPos,
currentPos));
// 2 observations for current target position
sensor.AddObservation(targetPos);
// 2 observations for movement velocity
sensor.AddObservation(rb.velocity.normalized);
// 1 observacion para mirar el modo ataque
sensor.AddObservation(attackMode);
}

```

Estadístiques llençades per Tensorboard de l'entrenament de la versió 2.3 amb 500000 steps.



Estadístiques llençades per Tensorboard de l'entrenament de la versió 2.3 amb el fitxer custom.



## 6 Conclusions i línies futures

Considero que el projecte ha estat un exit, s'han completat la majoria dels objectius, m'ha servit per tenir més clar en lo que es basa el Deep Learning i el Reinforcement Learning i finalment m'ha permès treballar una mica més amb l'eina principal que em prencipí utilitzaré per fer el TFG.

Dic que s'han comples gairabe el cent per cent dels objectius perquè, una de les coses que tenia pensat d'implementar però per questions tècniques referents a errors no vaig poder va ser que l'Agent quan detecti un enemic, canvi el seu target cap a ell. Tot i que l'Agent quan detecta un enemic es possa en mode atac i si l'enemic el toca a ell es mor, l'Agent continua amb el seu target de limits, però no fa una intensió real de dirigir-se al enemic proper.

Aquesta ampliació de la modelització seria una de les possibles línies futures del projecte.

Si estas interessat en l'implementació del projecte, et convido a visitar el meu repositori en Github, a les altures d'haver entregat el projecte, el repositori hauria d'estar public.

## 7 Referències

### Referències

- [1] Unity ml-agents documentation <https://github.com/Unity-Technologies/ml-agents/tree/main/docs>.
- [2] Understand machine learning plot results <https://medium.com/aureliantactics/understanding-ppo-plots-in-tensorboard-cbc3199b9ba2>.