Aprenentatge per reforç multiagent en un entorn virtual

Estudiant

Xavi Linares Dalmau

Màster d'Enginyeria Informàtica

Àrea d'Intel·ligència Artificial

Director

Samir Kanaan Izquierdo

Professor responsable de l'assignatura

Carles Ventura Royo

Data de lliurament: 23/06/2020

Universitat Oberta de Catalunya

AGRAÏMENTS

Als experts divulgadors Carlos Santana, per ser una font d'inspiració i motivació,
i Adam Kelly per els imprescindibles tutorials de ML-Agents publicats a Immersive Limit.

A la meva família per la paciència i l'acompanyament en les dificultats.

Als amics per fer més fàcil el camí,
especialment Jesús per la implicació amb els dissenys artístics.

A l'equip docent pel seu suport, consells i adaptació a la situació excepcional.

FITXA DEL TREBALL FINAL

Títol del treball	Aprenentatge per reforç multiagent en un entorn virtual
Nom de l'autor	Xavi Linares Dalmau
Nom del professor col·laborador	Samir Kanaan Izquierdo
Nom del professor responsable de l'assignatura	Carles Ventura Royo
Data de lliurament	23/06/2020
Titulació o programa	Màster d'Enginyeria Informàtica
Àrea del Treball Final	Intel·ligència Artificial
Idioma del treball	Català
Paraules clau	*artificial intelligence, machine learning, multi-agent reinforcement learning*

Resum del Treball (màxim 250 paraules)

En aquest treball de l'àmbit de la intel·ligència artificial, concretament de la branca de l'aprenentatge automàtic, volem estudiar les tècniques d'aprenentatge per reforç en un entorn virtual amb diversos agents que tenen una tasca competitiva.

L'estudi consisteix en la creació d'un entorn virtual d'aprenentatge mitjançant l'eina Unity, on implementem aprenentatge per reforç amb el complement Unity Machine Learning Agents (ML-Agents) i TensorFlow, i n'analitzem els resultats. Seguint aquest procés de manera iterativa, realitzem tres modificacions de l'entorn d'aprenentatge per observar com s'adapta el procés d'aprenentatge per reforç a cada situació.

Els resultats del projecte evidencien el potencial de l'aplicació de tècniques d'aprenentatge per reforç per tal d'optimitzar els processos automàtics, fins a límits que van més enllà de la capacitat de programació humana. Per altra banda, també identifiquem les dificultats que ens anem trobant durant l'aplicació d'aquestes tècniques i com les anem solucionant.

Abstract (in English, 250 words or less)

In this work in the field of artificial intelligence, specifically in the field of machine learning, we want to study the techniques of reinforcement learning in a virtual multi-agent competitive environment.

The study consists of the creation of a virtual learning environment using Unity, then we implement reinforcement learning with the Unity Machine Learning Agents (ML-Agents) and TensorFlow add-ons, and analyze the results. Following this process iteratively, we make three modifications to the learning environment to observe how the reinforcement learning process is adapted to each situation.

The results of the project highlight the potential application of reinforcement learning techniques in order to optimize automatic processes, to limits that go beyond human programming ability. On the other hand, we also identify the difficulties we encounter during the application of these techniques and how we solve them.

Índex

Llista de figures

Taules

Il·lustracions

1. Introducció

1.1 Context i justificació del Treball

En l'actualitat la intel·ligència artificial és present en gran part dels dispositius que ens rodegen i que fem servir en el dia a dia, des de la complexitat de les aplicacions que podem tenir en el nostre telèfon intel·ligent que fins al senzill[1] robot aspirador que tenim a casa i a qui tantes hores devem.

És per tant un tema d'interès general el fet de seguir millorant els algoritmes d'intel·ligència artificial que empren tots aquests dispositius, cada vegada més nombrosos i interconnectats entre ells gràcies a l'explotació de tecnologies com el 5G (el que coneixem com a internet de les coses). La optimització dels mecanismes de decisió no només ha de permetre el funcionament d'aquests sistemes amb un abast en constant creixement, sinó que també l'escenari climàtic i ecològic del planeta ens obliga a que ho facin de manera energèticament eficient i sostenible. A tall d'exemple, el robot aspirador que actualment és capaç de trobar el camí per recórrer la totalitat d'una superfície sortejant obstacles, amb nous sensors podria aprendre amb quina freqüència aquesta superfície ha de ser netejada i ser més eficient.

En aquest treball final pretenem explorar la branca d'aprenentatge màquina de la intel·ligència artificial, posant en pràctica mètodes d'aprenentatge per reforç en un entorn virtual on diversos agents interactuen de manera competitiva. Volem aprendre com es realitzen aquest tipus d'entrenaments i analitzar-ne els resultats. Seguint amb l'exemple anterior, si entrenem el model d'intel·ligència artificial del robot aspirador incorporar un agent que li dificulti la tasca, serà aquest més eficient alhora d'evitar obstacles (com ara mascotes) i trobar la ruta més adient?

Prenem com a referència o inspiració per a aquest treball un estudi recent publicat a OpenAI[2], on aplicant aprenentatge per reforç en un entorn virtual d'un joc simple (fet i amagar), els agents acaben desenvolupant comportaments intel·ligents molt complexes i fins i tot inesperats pels científics responsables.

Dit això, la finalitat del treball és la d'explorar la implementació d'aquestes tècniques d'aprenentatge màquina per mitjà de posar-les en pràctica en un entorn virtual.

[1] Robots at the tipping point: the road to iRobot Roomba [Data de consulta 23/06/2020] <https://ieeexplore.ieee.org/document/1598056>
[2] Emergent Tool Use from Multi-Agent Interaction [Data de consulta 26/06/2020] <https://openai.com/blog/emergent-tool-use/>

1.2 Objectius del Treball

Definim els objectius del treball amb una lògica progressiva, alineats amb les diferents fites que necessitem assolir per a la finalitat del projecte que és la d'analitzar l'aplicació de tècniques d'aprenentatge automàtic per reforç en un entorn multi-agent.

Per tant, identifiquem els objectius generals com a grans fases amb un producte resultant i necessari per avançar a la següent, i cadascun d'ells el desglossem en una sèrie d'objectius més específics.

El primer objectiu general té a veure amb la necessitat de disposar de disposar de l'entorn on poder aplicar aprenentatge per reforç (*reinforcement learning, RL*). Es tracta d'un entorn virtual que dissenyem i desenvolupem per a aquesta finalitat concreta, i en els objectius específics determinem com fer-ho:

OBJ1. Dissenyar i desenvolupar l'entorn virtual amb tots els elements i mecanismes necessaris per a poder aplicar l'entrenament als agents.
OBJ1.1. Instal·lar i configurar l'entorn de desenvolupament amb què construirem el nostre entorn de simulació.
OBJ1.2. Dissenyar i desenvolupar l'escenari amb els diferents objectes que el formen i els algoritmes que defineixen la seva creació i interacció.
OBJ1.3. Dissenyar i desenvolupar els agents com a objecte i amb les seves mecàniques d'interacció amb l'entorn.
OBJ1.4. Verificar l'adequació entre l'escenari i els agents.

Taula 1. Definició del primer objectiu general.

Un cop en disposició de l'entorn d'entrenament, ja hi podem implementar l'aprenentatge per reforç i analitzar-ne els resultats, i aquest és el segon objectiu general. Desgranem aquest objectiu en necessitats més concretes i en surten els objectius específics:

OBJ2. Aplicar aprenentatge per reforç als agents i observar els resultats.
OBJ2.1. Integrar l'eina per aplicar RL als agents a l'entorn de desenvolupament.
OBJ2.2. Dissenyar i desenvolupar la implementació de RL dels agents.
OBJ2.3. Verificar els resultats de la implementació RL sobre l'entorn.

Taula 2. Definició del segon objectiu general.

Finalment, volem realitzar iteracions per afegir complexitat a l'entorn de simulació i poder comparar la resposta de l'entrenament per reforç en les diferents casuístiques. És doncs, el últim objectiu general. Els d'objectius específics depenen del nombre d'iteracions que pretenem realitzar, i en són tres:

OBJ3. Analitzar els resultats de l'aplicació de RL incrementant la complexitat de l'entorn virtual
OBJ3.1. Dissenyar i desenvolupar la primera modificació de l'entorn de simulació amb l'adequació de l'algoritme de RL.
OBJ3.2. Verificar els resultats de la implementació RL sobre l'entorn modificat amb la primera iteració.
OBJ3.3. Dissenyar i desenvolupar la segona iteració de l'entorn de simulació amb l'adequació de l'algoritme de RL.
OBJ3.4. Verificar els resultats de la implementació RL sobre l'entorn modificat amb la segona iteració.
OBJ3.5. Dissenyar i desenvolupar la tercera iteració de l'entorn de simulació amb l'adequació de l'algoritme de R
OBJ3.6. Verificar els resultats de la implementació RL sobre l'entorn modificat amb la tercera iteració.

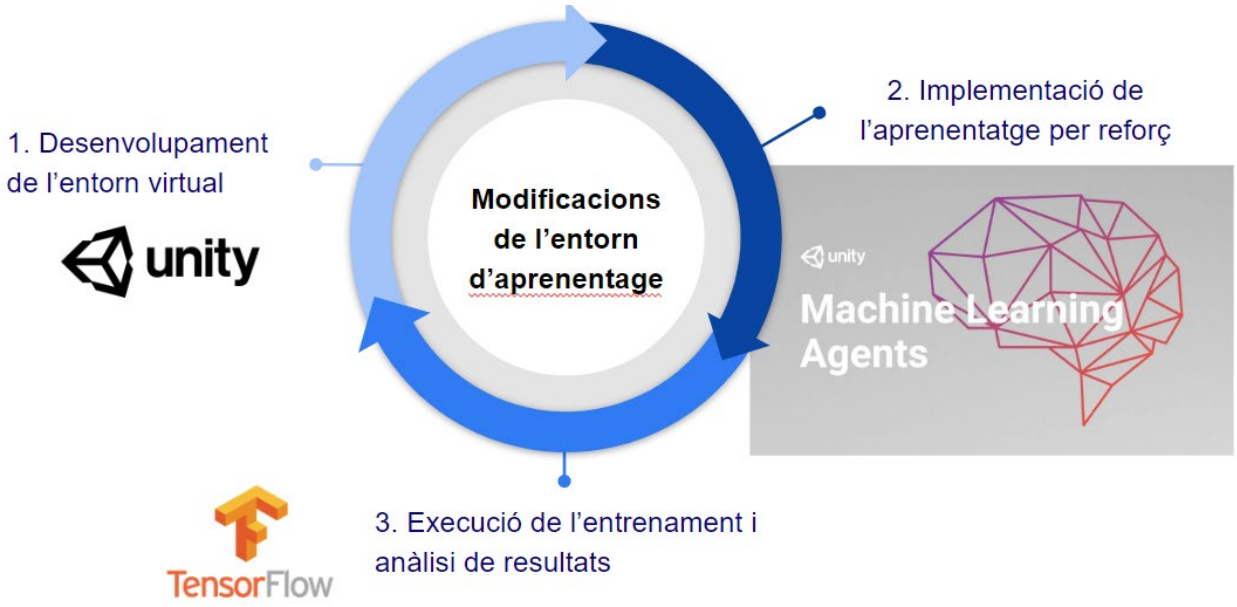
Taula 3. Definició del tercer objectiu general.

1.3 Enfocament i mètode seguit

Per posar en pràctica l'aprenentatge per reforç identifiquem dos escenaris possibles referents a l'entorn d'aplicació. Una opció és un entorn electrònic on els agents siguin robots amb una sèrie de sensors, actuadors i un processador on implementar l'algoritme d'aprenentatge. La segona és dur a terme l'aprenentatge de manera virtual en un entorn de simulació 3D on desenvolupar tant l'escenari com els agents i l'algoritme d'aprenentatge.

L'opció virtual és més viable que la electrònica perquè tenim a l'abast i de manera més immediata els recursos necessaris per executar el treball, així com també està més alineada amb els coneixements adquirits durant el màster d'enginyeria informàtica.

En relació amb el mètode seguit, tal com es planteja en els objectius, seguim un enfoc iteratiu on les passes a seguir són, en aquest ordre: el desenvolupament de l'entorn virtual, la implementació de l'aprenentatge per reforç, i l'execució de l'entrenament amb el corresponent anàlisi de resultats. De tal manera que en la primera iteració generem i experimentem amb un entorn base, sobre el qual en les tres iteracions posteriors realitzem modificacions per anar aplicant i observant l'aprenentatge dels agents.



Il·lustració 1. Esquema de la metodologia emprada.

Per a desenvolupar l'entorn virtual fem servir l'entorn de desenvolupament 3D Unity[3], i el seu complement Unity Machine Learning Agents Toolkit (ML-Agents)[4] per a l'aplicació d'aprenentatge per reforç, ja que això ens facilita enormement la integració entre entorn i sistema d'aprenentatge. Per a executar els entrenaments, ML-Agents fa ús de les llibreries de TensorFlow[5] i això ens permet analitzar els resultats amb el visor TensorBoard.

1.4 Planificació del Treball

A l'hora d'elaborar la planificació prenem com a referència les fites definides pels lliuraments de les proves d'avaluació contínua (PAC), que ens permeten establir dues grans fases, relacionar-les amb els objectius i detallar les tasques necessàries per assolir-los.

Tasca	Objectiu
Pla de treball - PAC1 Lliurament: 29 de març de 2020	-
Elaboració del pla de treball	-
Fase 1 - PAC 2 Lliurament: 20 de maig de 2020	**OBJ1: Dissenyar i desenvolupar l'entorn virtual amb tots els elements i mecanismes necessaris per a poder aplicar l'entrenament sobre els agents.**
Instal·lació i training Unity	OBJ1.1. Instal·lar i configurar l'entorn de desenvolupament amb què construirem el nostre entorn de simulació.
Desenvolupament entorn virtual	OBJ1.2. Dissenyar i desenvolupar l'escenari amb els diferents objectes que el formen i els algoritmes que defineixen la seva creació i interacció.
Desenvolupament agents	OBJ1.3. Dissenyar i desenvolupar els agents com a objecte i amb les seves mecàniques d'interacció amb l'entorn.

[3] Unity Real-Time Development Platform [Data de consulta 23/06/2020] <https://unity.com/>
[4] Unity Machine Learning Agents Toolkit [Data de consulta 23/06/2020] <https://unity3d.ai/>
[5] TensorFlow [Data de consulta 23/06/2020] <https://www.tensorflow.org/>

Proves i adequacions	OBJ1.4. Verificar l'adequació entre l'escenari i els agents.
Elaboració informe de seguiment	-
Fase 2 - PAC 3 Lliurament: 1 de juny de 2020	**OBJ2: Aplicar aprenentatge per reforç als agents i observar els resultats** **OBJ3. Analitzar els resultats de l'aplicació de RL incrementant la complexitat de l'entorn virtual**
Instal·lació i training Unity ML	OBJ2.1. Integrar l'eina per aplicar RL als agents a l'entorn de desenvolupament.
Disseny estratègia RL	OBJ2.2. Dissenyar i desenvolupar la implementació de RL dels agents.
Desenvolupament algoritme RL	
Proves i adequacions	OBJ2.3. Verificar els resultats de la implementació RL sobre l'entorn.
Desenvolupament iteració 1	OBJ3.1. Dissenyar i desenvolupar la primera modificació de l'entorn de simulació amb l'adequació de l'algoritme de RL.
Proves i adequacions iteració 1	OBJ3.2. Verificar els resultats de la implementació RL sobre l'entorn modificat amb la primera iteració.
Desenvolupament iteració 2	OBJ3.3. Dissenyar i desenvolupar la segona iteració de l'entorn de simulació amb l'adequació de l'algoritme de RL.
Proves i adequacions iteració 2	OBJ3.4. Verificar els resultats de la implementació RL sobre l'entorn modificat amb la segona iteració.
Desenvolupament iteració 2	OBJ3.3. Dissenyar i desenvolupar la tercera iteració de l'entorn de simulació amb l'adequació de l'algoritme de RL.
Proves i adequacions iteració 3	OBJ3.4. Verificar els resultats de la implementació RL sobre l'entorn modificat amb la tercera iteració.
Aplicar models gràfics	-

Memòria - PAC 4 **Lliurament: 16 de juny de 2020**	-
Redacció de la memòria	-
Revisió de la memòria	-
Presentació - PAC 5a **Lliurament: 21 de juny de 2020**	-
Redacció de la presentació	-
Enregistrament de la presentació	
Defensa - PAC 5b **Del 25 al 28 de juny de 2020**	
Respondre a les preguntes	

Taula 4. Relació de tasques i objectius.

Per establir la temporalització de la planificació ens basem en les fites de lliurament de les PAC de seguiment del projecte, i anem ajustant la dedicació en dies per a cadascuna de les tasques.

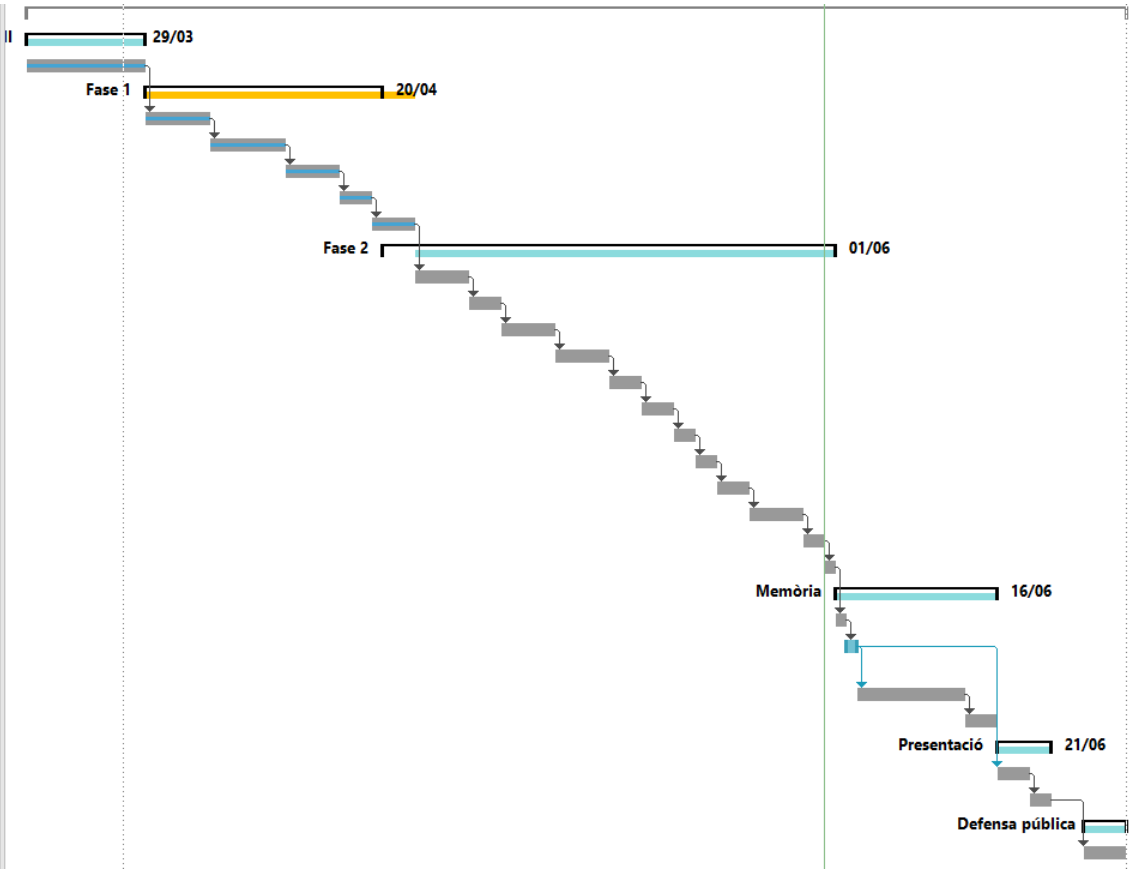
A nivell de recursos la dedicació és d'una persona a temps parcial durant tots els dies de la setmana. Tenint en compte que la dedicació estimada per al treball final de 12 crèdits ECTS és de 300 hores, amb una durada total de 92 dies, la dedicació és d'aproximadament 3 hores al dia.

Com a fet rellevant que s'ha produït durant el projecte i ha impactat de ple a la planificació, destacar la situació de crisis sanitària ocasionada per la pandèmia del COVID-19, amb un estat d'alarma[6] del 14 de març fins al 21 de juny de 2020, per el qual hem tingut un escenari de confinament total de diverses setmanes, tancament d'escoles, teletreball generalitzat. Aquest canvi dràstic de context ha generat un procés d'adaptació i una dedicació més variable d'hores a les tasques del projecte. Al mateix temps i atenent a la situació, des de l'equip docent s'ha concedit una ampliació dels terminis de lliurament de les posteriors entregues que han possibilitat la finalització de les mateixes. És per aquest motiu que finalment la durada del treball final és de 102 dies en comptes dels 92 calculats inicialment.

[6] Real Decreto 463/2020, de 14 de marzo, por el que se declara el estado de alarma para la gestión de la situación de crisis sanitaria ocasionada por el COVID-19. [Data de consulta 23/06/2020] <https://www.boe.es/eli/es/rd/2020/03/14/463>

A continuació mostrem el diagrama de Gantt amb la darrera revisió de la planificació temporal:

TFM	102 días
Pla de treball	11 días
Elaboració pla de treball	11 días
Fase 1	22 días
Instal·lació i training Unity	6 días
Desenvolupament entorn virtua	7 días
Desenvolupament agents	5 días
Proves i adeqüacions	3 días
Elaboració informe de seguimer	4 días
Fase 2	42 días
Instal·lació i training Unity ML	5 días
Disseny estratègia RL	3 días
Desenvolupament algoritme RL	5 días
Proves i adeqüacions	5 días
Desenvolupament iteració 1	3 días
Proves i adeqüacions iteració 1	3 días
Desenvolupament iteració 2	2 días
Proves i adeqüacions iteració 2	2 días
Desenvolupament iteració 3	3 días
Proves i adeqüacions iteració 3	5 días
Aplicar models gràfics	2 días
Elaboració informe de seguimer	1 día
Memòria	15 días
Netejar arxius de codi font	1 día
Desplegament escenaris en format web	1 día
Redacció de la memòria	10 días
Revisió de la memòria	3 días
Presentació	5 días
Redacció de la presentació	3 días
Enregistrament de la presentaci	2 días
Defensa pública	4 días
Respondre a les preguntes	4 días



Il·lustració 2. Diagrama de Gantt (darrera revisió l'1 de juny de 2020)

1.5 Breu sumari de productes obtinguts

Com a productes resultants a la finalització del treball, tenim:

- 4 escenaris virtuals on hem implementat aprenentatge per reforç i estudiat els resultats
- 7 models neuronals entrenats
- Demostració interactiva
- Pla de treball
- 2 informes de seguiment del treball
- Memòria del treball
- Presentació del treball

1.6 Breu descripció dels altres capítols de la memòria

Aquest treball s'estructura en 3 grans capítols:

- En el capítol 2, titulat "**L'entorn virtual de simulació**" expliquem l'eina de desenvolupament que hem utilitzat per desenvolupar l'entorn virtual, quins han estat els criteris de disseny alhora de conceptualitzar-lo, i finalment detallem de quina manera l'hem desenvolupat analitzant cadascun dels objectes que en formen part.

- En el capítol 3, "**Aplicació de l'aprenentatge per reforç**", establim el marc teòric de referència del treball i de les tècniques que utilitzarem i expliquem quines eines hem emprat per implementar l'aprenentatge. Seguidament detallem les característiques del disseny de l'aprenentage i com les implementem a l'entorn virtual, element per element. Finalment relatem tota la consecució d'entrenaments realitzats, les dificultats obtingudes que ens porten a dur a terme ajustos de la implementació, i els resultats obtinguts en cada cas així com els models obtinguts.

- En el capítol 4, "**Iteracions de l'entorn d'aprenentatge**", detallarem les tres iteracions de l'entorn estudiat en els capítols anteriors, quina modificació introdueix cadascuna de les iteracions, com s'implementa, quina finalitat persegueix, i quins són els resultats obtinguts en els diferents entrenaments. En aquest capítol també s'inclou una incorporació d'elements artístics a l'entorn virtual, i una l'explicació de com hem dut a terme la publicació del projecte en un format web interactiu i disponible a través d'Internet.

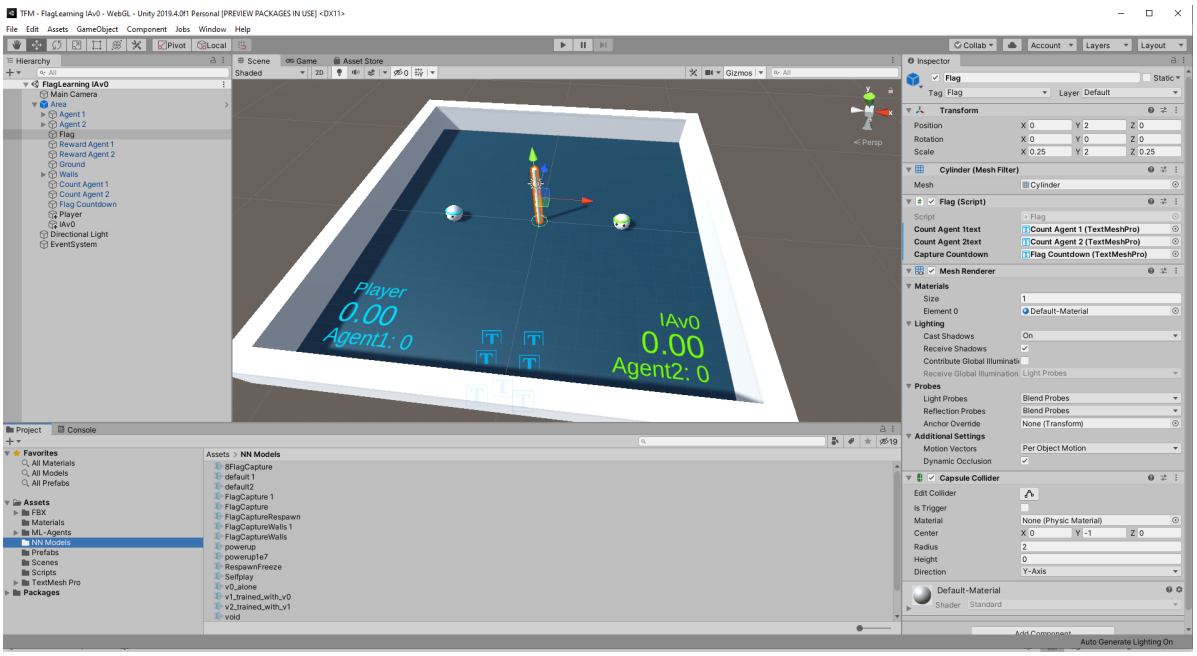
2. L'entorn virtual de simulació

En aquest apartat ens centrem en l'escenari virtual obtingut en la primera fase del treball, quines eines hem fet servir per desenvolupar-lo, quines premisses de disseny hem seguit, i els elements que el composen.

2.1 Entorn de desenvolupament Unity

Tal com hem vist a la introducció, l'entorn de desenvolupament utilitzat és Unity. Es tracta d'una eina àmpliament utilitzada per a la producció de videojocs i força popular, de tal manera que tenim a l'abast una base de coneixement en forma de documentació, recursos d'autoaprenentatge, i de comunitats d'usuaris, que ens facilita l'aprenentatge en el seu ús.

Unity disposa d'una interfície gràfica que ens permet dur a terme la composició dels objectes en l'escena, editar-ne les propietats, programar el seu comportament en llenguatge C# (C Sharp), i executar l'escena, entre moltes altres funcionalitats.



Il·lustració 3. Interfície d'usuari de Unity

Un altre avantatge de Unity són els complements que es poden integrar per disposar de noves funcions, com ara el que ens permetrà aplicar aprenentatge per reforç als agents.

D'aquest entorn de desenvolupament també ens interessa la capacitat d'exportar l'escena optimitzada per a diferents plataformes com ara web, PC o dispositius mòbils. Així, podem facilitar una demostració web interactiva dels escenaris implementats.

La versió de Unity més actualitzada en el moment de lliurar aquest treball és la 2019.4.0.

Quant als scripts que podem programar, Unity ens proporciona per defecte alguns mètodes que permeten controlar el flux d'execució en base a la freqüència d'actualització dels fotogrames de la pantalla (framerate) i a una freqüència d'actualització regular:

Mètode	Descripció
Start()	S'executa només començar, abans de l'actualització del primer fotograma.
Update()	S'executa cada vegada que s'actualitza un fotograma (framerate).
FixedUpdate()	S'executa en intervals de temps regulars.

Taula 5. Funcions predeterminades de Unity.

Per saber a quina de les funcions anteriors és més adient col·locar una determinada instrucció, hem de tenir en compte que la *framerate* és variable i oscil·la entre els 30 i 60 fotogrames per segon i la freqüència d'actualització regular sempre serà constant i molt més ràpida (per exemple 200 vegades per segon).

Així doncs, comencem incorporant la inicialització de totes les variables i objectes necessaris en el mètode *Start()*. Dins *Update()* posem les ordres que no necessiten una consistència temporal molt estricta com les actualitzacions d'interfície d'usuari, textos, moviments d'objectes sense implicacions físiques o el processat de les entrades de teclat i ratolí. Per altra banda, dins de *FixedUpdate()* incorporem les accions que sí requereixen una coherència temporal com són les que afecten a les propietats físiques dels objectes en escena. D'aquesta manera, els efectes físics de l'escena es van calculant i representant "en rerefons", i anem mostrant per pantalla fotografies de l'escena amb una freqüència inferior però suficient perquè no es notin els talls a nivell de la percepció humana, obtenint així una visualització fluïda.

2.2 Disseny de l'escenari inicial

Per al de poder aplicar aprenentatge per reforç en els termes descrits en els capítols anteriors, en el disseny de l'escenari de simulació hem tingut en compte les següents premisses:

- Ha d'admetre modificacions posteriors sense fer grans canvis.
- L'objectiu a acomplir pels agents ha de ser senzill i directe.
- La tasca dels agents ha de ser visualment clara per a l'espectador.
- Els agents han de competir per fer la tasca, no cooperar.

Tenint en compte això, l'elecció és implementar un joc de captura de bandera, on l'objectiu dels agents és tocar una bandera col·locada a la mateixa superfície on ells es desplacen, i només l'agent que la captura primer guanya un punt.

Per tant, els objectes que tenim en aquest escenari inicial, i les seves mecàniques, són les següents:

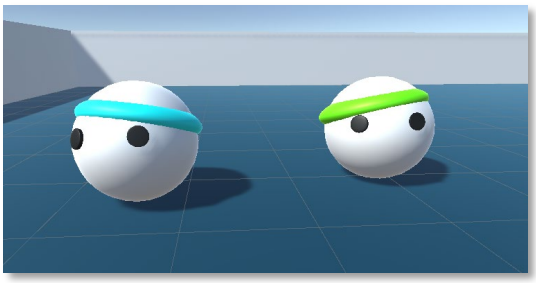
Objecte	Mecànica
Agent	Desplaçament sobre la superfície de joc: endavant, endarrere i gir (rotació sobre l'eix vertical).
Bandera	En entrar en contacte amb un agent, canvia al color d'aquest agent, i inicia un compte enrere de 3 segons. Passat aquest temps, suma un punt a l'agent que l'ha capturat, es reinicialitza i es mou a una ubicació nova. Si un agent l'està capturant i és col·lisionada per l'altre, atura el compte enrere i inicia un de nou, per al nou agent.

Il·lustració 4. Disseny dels objectes de l'escenari

2.2 Desenvolupament de l'escenari inicial

A continuació detallem objecte per objecte com hem dut a terme la construcció de l'escenari:

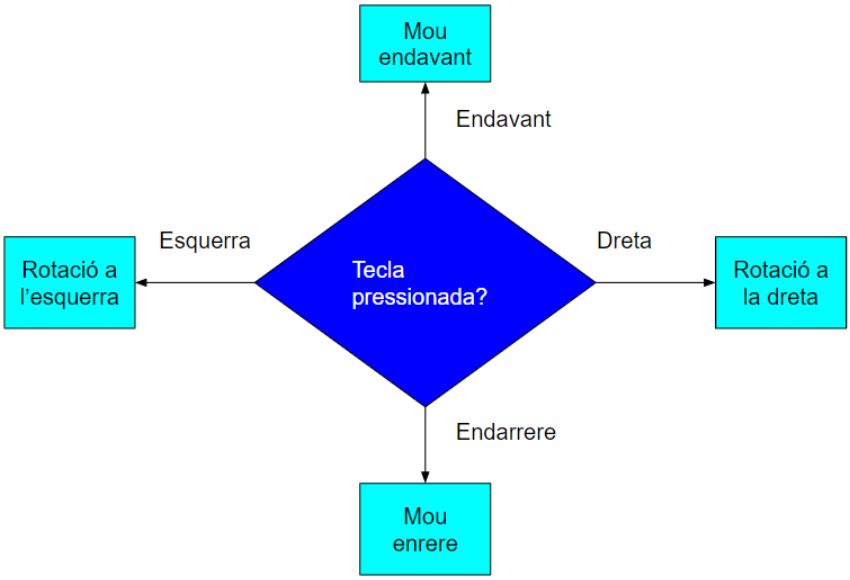
2.2.1 Agent



Il·lustració 5. Model dels agents.

Els agents tenen un cos de forma esfèrica, amb un parell d'ulls per senyalitzar la direcció cap a on s'enfoquen, i una bandana que permet identificar el seu color.

En quant a la programació, en aquesta fase només ens cal implementar el desplaçament, seguint el flux que es mostra a continuació:

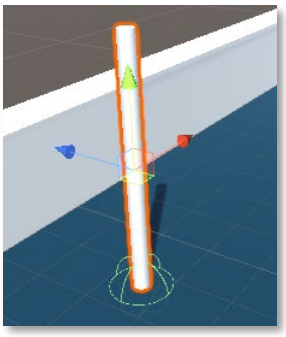


Il·lustració 6. Diagrama de moviment de l'agent.

Destacar que per a les accions de desplaçament endavant i enrere fem ús del mètode *addForce* per dotar d'inèrcia al moviment de l'agent i pugui col·lisionar amb el rival.

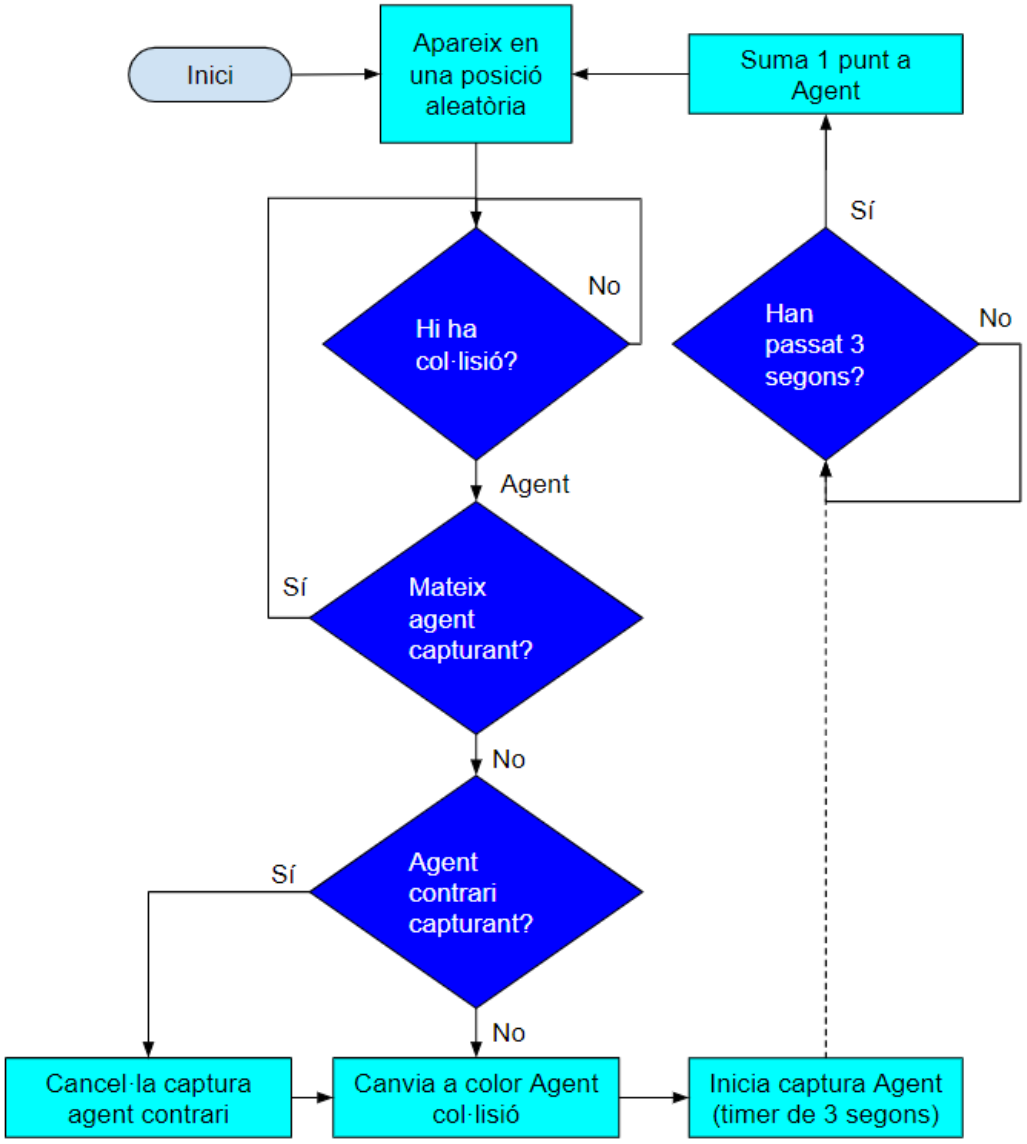
2.2.2 Bandera

La bandera té forma de cilindre i és de color blanc, excepte quan està sent capturada per un agent, que adquireix el color corresponent.



Il·lustració 7. Model de la bandera.

El mecanisme de captura recau sobre la bandera, com es pot veure en el diagrama:



Il·lustració 8. Diagrama de captura de la bandera.

2.2.3 Comptadors

Per poder visualitzar d'una manera directa l'avanç i els resultats d'un entrenament, afegim uns elements *TextMeshPro*[7] inserits en el propi escenari que ens permeten mostrar per a cada agent la següent informació en temps d'execució:

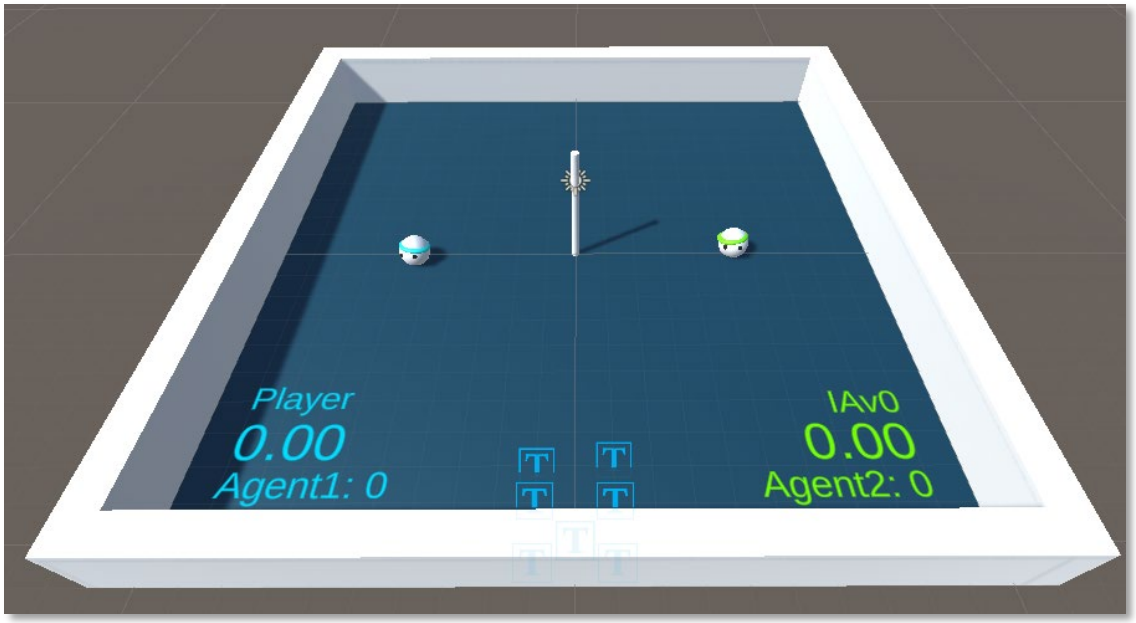
- Nom del model neuronal aplicat (o que s'està entrenant) sobre l'agent. O "player" en cas que no tingui aplicat un model i sigui controlat per un jugador humà.
- El valor de la recompensa acumulada.
- El nombre de captures de bandera acumulades.

Il·lustració 9.Comptadors.

2.2.4. Elements passius

Com a elements estàtics disposem de la superfície sobre la que la resta d'elements es recolzen, consistent en un pla d'amplada i llargada finites, amb una sèrie de murs rectangulars en el seu perímetre. Aquestes parets bloquegen el moviment dels agents més enllà el límit de la superfície.

A la següent imatge podem visualitzar el muntatge de l'escena complerta:

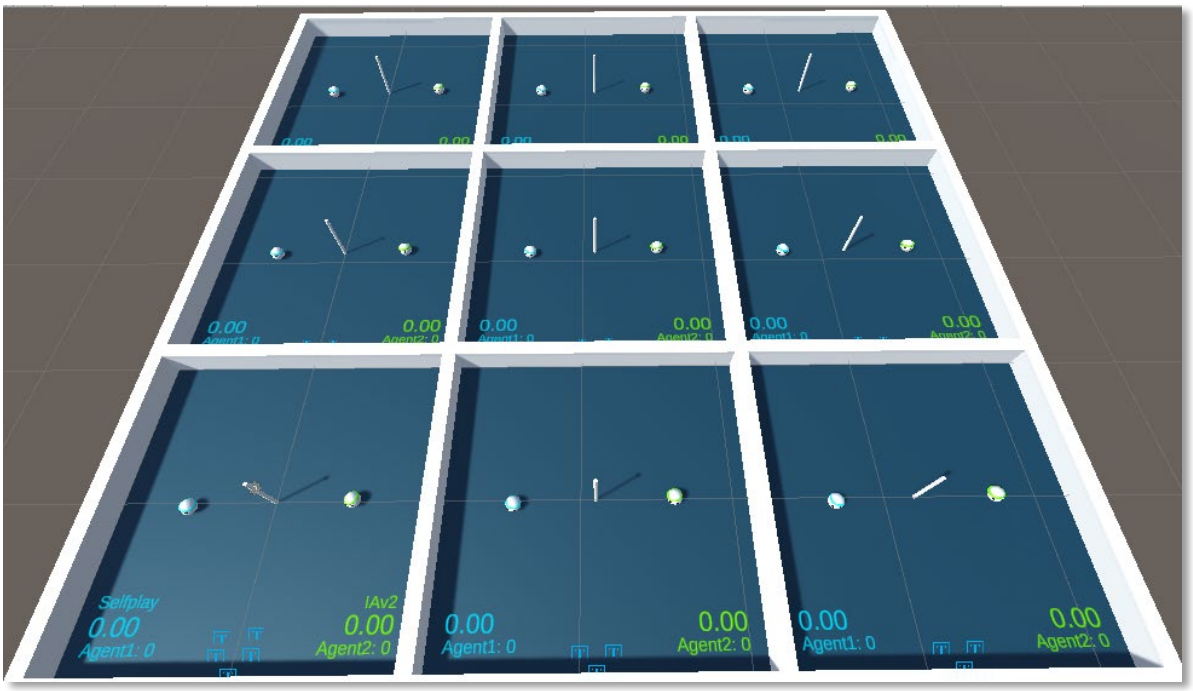


Il·lustració 10. Escenari inicial complert.

[7] TextMesh Pro [Data de consulta 23/06/20] <https://docs.unity3d.com/Manual/com.unity.textmeshpro.html>

2.2.5 Àrea

L'àrea és un objecte abstracte que ens permet agrupar la resta dels objectes mencionats en una única entitat (o *Prefab*), que ens facilita el seu reaprofitament i manteniment. Per exemple, podem instanciar dins el mateix escenari vàries rèpliques de l'àrea que funcionin de manera independent unes de les altres sense modificar els seus components interns.



Il·lustració 11.Diverses àrees en paral·lel.

Tal com veurem més endavant, la paral·lelització dels entrenaments mitjançant múltiples àrees és molt útil per tal d'obtenir models més precisos en menys temps i per tant guanyar eficiència.

3. Aplicació de l'aprenentatge per reforç

En aquest capítol entrem ja a detallar com dissenyem, implementem i apliquem l'aprenentatge per reforç a l'entorn que hem desenvolupat. També detallem les dificultats trobades i els resultats obtinguts.

3.1. Conceptes

A continuació introduïm els conceptes de l'àmbit de la intel·ligència artificial rellevants per a la finalitat del treball, per tal de situar-nos en el marc teòric.

3.1.1 Intel·ligència artificial

La intel·ligència artificial (*Artificial Intelligence, AI*) és un camp de la ciència que té per finalitat la creació de sistemes que simulen processos d'intel·ligència. Entenent per intel·ligència la capacitat de percebre un entorn i prendre decisions que maximitzen la probabilitat d'assolir un objectiu concret.

Els elements que actuen en aquests sistemes s'anomenen agents.

La IA és un camp en constant evolució, amb moltes vessants d'opinió i branques d'especialització.

3.1.2 Aprenentatge màquina

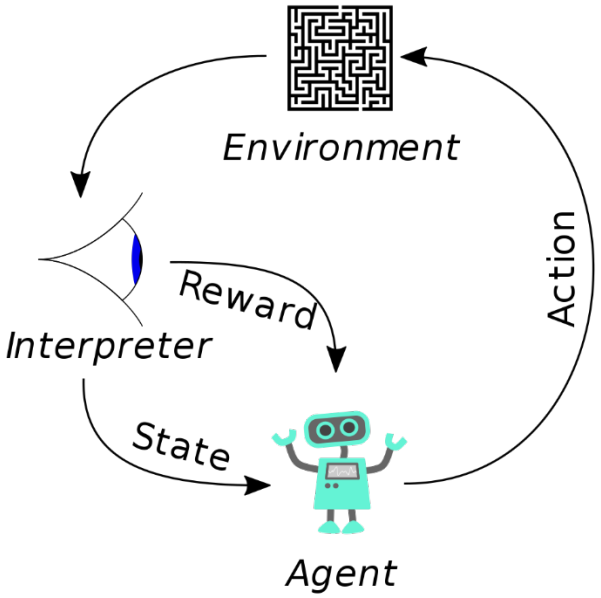
L'aprenentatge màquina (*Machine Learning, ML*) és la branca de la intel·ligència artificial on es recerquen tècniques per simular el pensament humà, concretament els processos de presa de decisions i d'aprenentatge

De l'aplicació de tècniques ML obtenim models computacionals que podem aplicar als agents perquè actuïn segons l'aprenentatge que representa. És a dir, no són models programats per una persona humana, sinó que han estat entrenats de manera automàtica mitjançant un algoritme d'aprenentatge.

Les xarxes neuronals són models computacionals inspirats en el model biològic humà i són el tipus de model que generem en els entrenaments amb ML-Agents, que posteriorment apliquem als agents.

3.1.3 Aprenentatge per reforç

L'aprenentatge per reforç (*Reinforcement Learning, RL*) fa referència a aquelles tècniques de l'àmbit de l'aprenentatge automàtic que consisteixen en un agent actuant sobre un entorn del qual en percep unes observacions i recompenses, i que té per finalitat maximitzar aquesta recompensa.



Il·lustració 12. Aprenentatge per reforç. Font: Wikipedia.

D'aquesta manera, l'agent comença a realitzar accions de manera aleatòria i en base al retorn que obté en forma de recompensa i les observacions que percep, correlaciona "aprèn" quines accions li repercutiran en una màxima recompensa a cada moment.

La recompensa és gradual i positiva o negativa, de manera que permet establir una relació directa amb la tasca a assolir i amb els comportaments que volem potenciar o evitar.

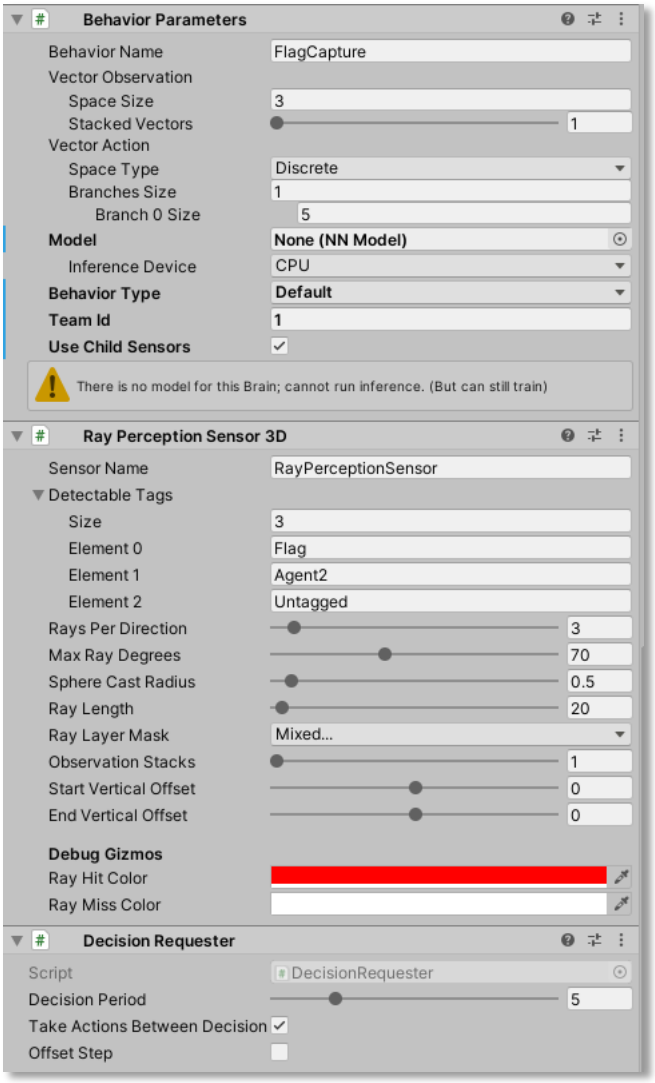
Aquesta és la metodologia que aplicarem en el nostre entorn d'aprenentatge, tal com veurem més endavant.

3.1.4 Curriculum learning

El curriculum learning és una configuració d'aprenentatge per reforç que consisteix en incrementar gradualment la dificultat de la tasca a realitzar conforme l'agent va millorant en la recompensa obtinguda. Els llindars de recompensa que desencadenen un augment de la dificultat de la tasca s'anomenen lliçons.

3.2 L'eina Unity Machine Learning Agents (ML-Agents)

El programari que fem servir per implementar l'aprenentatge per reforç en el nostre sistema és Unity Machine Learning Agents (ML-Agents). És un projecte de codi obert impulsat per Unity Technologies, que s'instal·la com a *plugin* de l'entorn de desenvolupament Unity i ens permet incorporar de manera senzilla els components i les funcions per dur a terme RL, com per exemple:



Il·lustració 13. Components ML-Agents aplicats a l'agent.

A la imatge veiem de manera gràfica els components *Behavior Parameters*, *Ray Perception Sensor 3D* i *Decision Requester,* que apliquen sobre els objectes Agent i que més endavant veurem. Amb el paràmetre *Behavior Type* determinem si l'agent farà servir un model ja entrenat (el seleccionat a *Model*), si el controlarem de forma heurística, o si l'entrenarem.

Així doncs, ML-Agents gestiona a través de Unity totes les accions relacionades amb l'entorn que són recopilar les observacions, obtenir les recompenses i aplicar les accions, gestionar el cicle d'entrenament, i per aplicar els algoritmes de càlcul i de decisió es comunica amb la llibreria de codi obert TensorFlow mitjançant una API en Python.

Un cop finalitzat un entrenament obtenim un model neuronal que podem gestionar com a un objecte més a la interfície de Unity i aplicar-lo sobre els agents en mode inferència. En aquest mode l'agent ja no s'entrena i utilitza el model per a prendre les decisions.

3.3 TensorBoard

TensorBoard és una eina de visualització de dades que forma part del paquet TensorFlow i que ens permet observar estadístiques dels entrenaments realitzats de manera gràfica mitjançant uns taulers.

Ens permet observar dades molt útils per a interpretar com ha anat un entrenament, com les següents:

- Recompensa acumulada.
- Durada dels episodis.
- Ràtio d'aprenentatge.
- Variació de la política de decisió.

3.4 Disseny de l'aprenentatge

En aquesta secció determinem els diferents aspectes sobre com aplicarem l'aprenentatge, seguint les recomanacions de bones pràctiques[8] recomanades per els desenvolupadors de ML-Agents.

3.4.1 Episodis

Cada entrenament es composa d'una sèrie d'episodis on l'agent recull observacions, realitza accions i pren decisions per tal d'assolir un objectiu. Aquest objectiu s'identifica mitjançant la recompensa que l'agent obté com a resultat de les seves accions.

En primer lloc, concretem els criteris de finalització dels episodis:

- L'agent arriba a les 5000 accions.

[8] Agents [Data de consulta 23/06/20] <https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Learning-Environment-Design-Agents.md>

- L'agent completa la captura de la bandera.
- L'agent contrari completa la captura de la bandera.
- L'agent supera els límits de l'escenari (per efecte de la inèrcia salta el mur i cau al buit).

3.4.2 Recompenses

A continuació, necessitem determinar les recompenses que pot obtenir l'agent. Tenim en compte que la recompensa recomanada és entre +1 i -1. Ho veiem en la següent taula:

Esdeveniment	Recompensa
L'agent completa la captura de la bandera	+1
L'agent contrari completa la captura la bandera	-1
L'agent duu a terme una acció i no està capturant la bandera	-1 / 5000
L'agent supera els límits de l'escenari	-1

Taula 6. Esdeveniments i recompenses.

Amb aquesta política de recompenses pretenem que l'entrenament generi una correlació molt directa amb l'objectiu principal del joc que és capturar la bandera al donar la màxima recompensa en fer-ho, i per altra banda propiciar la rivalitat al donar la màxima recompensa negativa quan ho fa el contrari.

El fet de donar una petita recompensa negativa després de cada acció és per incentivar l'agent a completar l'objectiu el més ràpid possible ja que anirà acumulant una recompensa negativa proporcional al nombre d'accions, fins al límit de -1 que és quan finalitzarà l'episodi. Aquesta recompensa negativa s'atura si l'agent està en procés de captura de la bandera perquè no pot fer cap acció per accelerar-lo al tenir una durada fixa de 3 segons, i podria ser contraproduent. La superació dels límits de l'escenari genera la màxima recompensa negativa perquè es relacioni com a esdeveniment a evitar.

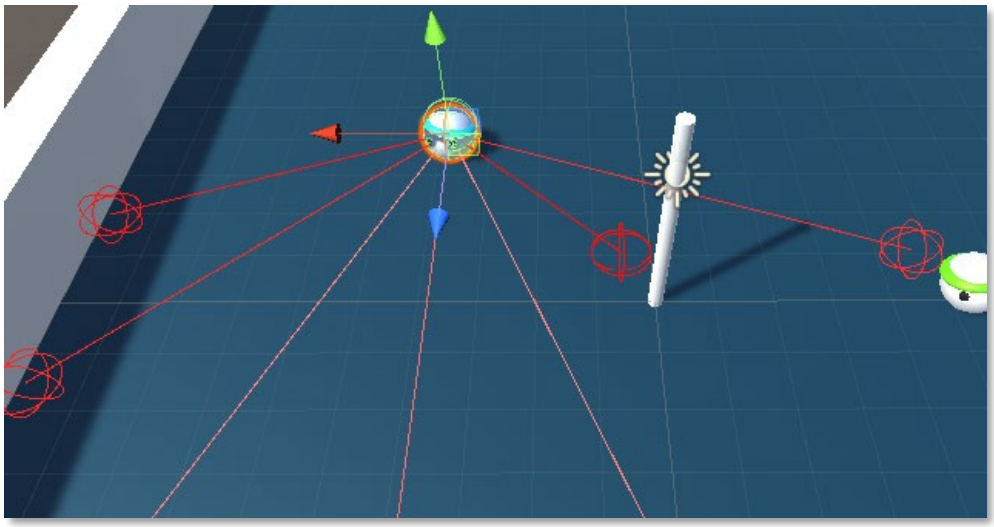
3.4.3 Observacions

Un cop establertes les recompenses podem identificar les observacions que recollirà l'agent del seu entorn i que li permetran prendre decisions. Hem de tenir en compte que com menys observacions més eficient serà l'entrenament pel fet de tenir menys variables, per tant han de ser poques i amb una finalitat rellevant. Les recollim a la següent taula indicant la finalitat de cadascuna d'elles:

Observació	Finalitat
Estat de la bandera (en captura per ell mateix, per l'agent contrari, o sense capturar).	Que l'agent adopti diferents comportaments segons l'estat de la bandera (defensar la captura, anar a capturar).
Velocitat de desplaçament de l'agent en l'eix x i en l'eix z.	Que l'agent acceleri o desacceleri tenint en compte que el seu moviment té inèrcia.
Sensor de percepció de rajos 3D. Identifiquem la posició de l'agent contrari, la bandera.	Que l'agent identifiqui els diferents elements de l'escenari per actuar en conseqüència.

Taula 7. Observacions.

Les dues primeres observacions les implementarem en el codi de l'agent i pel què fa al sensor de percepció de rajos 3D tant sols l'haurem d'incorporar com a component a l'agent. Tal com podem veure a la il·lustració, el component consisteix d'una sèrie de rajos projectats des del centre de l'objecte on s'incorpora, que en impactar amb un altre objecte l'identifiquen mitjançant l'atribut etiqueta.



Il·lustració 14. Sensor de percepció de rajos 3D.

El component permet configurar el nombre de rajos, l'amplitud en angles, la distància màxima d'impacte, el radi de l'esfera generada en col·lisionar i les etiquetes dels objectes que s'identificaran. Aquest mecanisme dotarà a l'agent de la informació espacial necessària per desplaçar-se per l'escenari localitzant l'objectiu i l'adversari.

3.4.4 Accions

Quant a les accions que l'agent pot dur a terme, són les que ja hem vist en la secció 2.2.1, referents al desplaçament. Són 5 possibilitats (rotar esquerra, rotar dreta, moure endavant, moure enrere, no fer res). Per tant el nostre vector d'accions tindrà una sola posició que pot prendre 5 valors enters diferents.

3.4.5 Hiperparàmetres de l'entrenament

Respecte de la parametrització de les característiques pròpies de la política d'entrenament (hiperparàmetres), farem ús de la configuració que per defecte estableix ML-Agents, ja que no és objecte d'aquest treball estudiar el funcionament dels algoritmes d'entrenament a un nivell més baix de detall.

En el fitxer trainer_config.yaml (veure annexos) tenim definits tots aquests paràmetres, anirem variant alguns d'ells durant l'execució dels entrenaments:

max_steps: el nombre d'episodis en què consisteix l'entrenament.

self_play: activa un mode d'entrenament on l'agent principal i el rival (diferenciats per un número *Team*) s'entrenen competint i alimentant el model de manera simultània. Al llarg de l'entrenament l'agent rival va realitzant canvis amb certa aleatorietat de la versió del model que estan entrenant i que s'infereix, per així introduir varietat i evitar el sobre-entrenament.

curiosity: habilita l'aplicació de recompenses intrínseques en base a els canvis produïts en les observacions de l'agent, és a dir, potencia un comportament d'explorar i descobrir.

Un darrer paràmetre de disseny que cal decidir és la ràtio de decisions que prendrà l'agent cada cert nombre d'accions. En aquest cas també seguim el criteri recomanat d'una decisió cada 5 accions.

3.4.6 Currículum learning

Acabant amb el disseny de l'aprenentatge, a l'escenari inicial aplicarem currículum learning amb la finalitat d'optimitzar el temps dels entrenaments i per tal d'observar els seus resultats.

Donat que en el nostre cas la tasca a assolir és capturar la bandera i impedir-ho a l'agent contrari, juguem amb dos factors: la distància màxima on la bandera pot col·locar-se després de ser capturada (per facilitar el procés d'aprenentatge de la captura), i la massa de l'agent (per induir l'habilitat d'empentar o apartar a l'agent contrari).

La mesura que prenem per avançar en el currículum és la recompensa mitjana que adquireix l'agent després de 100 episodis d'entrenament.

La configuració del currículum és la següent:

Llindar de recompensa	Distància màxima de la bandera (x i z)	Massa
0,0	0,0	20,0
0,2	2,5	17,5
0,4	5,0	15,0
0,6	7,5	12,5
0,8	9,5	10

Taula 8. Configuració del curriculum learning

Podem veure la parametrització detallada del currículum learning a l'annex: flag_capture.yaml

3.5 Desenvolupament dels algoritmes d'aprenentatge

Després de tenir dissenyat com volem realitzar l'aprenentatge, cal implementar els algoritmes necessaris per tal de dur-lo a terme

Més concretament, afegim lògica als següents objectes del nostre escenari, per tal que el seu comportament durant les sessions d'entrenament sigui el que hem definit:

3.5.1 Area

Tal com hem descrit en l'apartat 2.2.5 l'àrea és l'objecte agrupador de tota la resta i essencialment ens facilita l'accés a les propietats entre els seus objectes fills.

També implementa les següents funcions que permeten inicialitzar la posició dels objectes agents i bandera:

Funció	Descripció
PlaceFlag ():	Mou la bandera a una posició aleatòria de la superfície de joc. En cas de tenir actiu el currículum learning té en compte el valor de distància màxima en els eixos x i z que pertoqui.
PlaceAgent(int agentNum):	Mou l'agent especificat en el paràmetre "agentNum" a la seva posició inicial.

Taula 9. Funcions de l'àrea.

Per tant l'àrea inicialitza la posició dels agents i la bandera en el moment de començar un entrenament, i aquests mateixos objectes criden a aquestes funcions quan necessiten reinicialitzar la seva posició. Per exemple quan un agent cau de la superfície, o bé quan la bandera és capturada.

Addicionalment, l'àrea també actualitza el valor del comptador de la recompensa acumulada de cadascun dels agents, dins la funció *Update()*.

Disposem del codi d'aquesta versió de l'àrea a l'annex: Area.cs

3.5.2 Agent

L'agent implementa tota la lògica de la classe Agent de ML-Agents i per tant les funcions que controlen el cicle de realització dels entrenaments, que són les següents:

Funció	Descripció	Implementació
Initialize()	Similar al mètode *Start()*, és on realitzem les accions com inicialitzar les referències a altres objectes.	Assignem a la variable auxiliar *agentNum* el número corresponent a l'agent. Inicialitzem les referències als objectes àrea i bandera.

OnEpisodeBegin()	Es crida cada vegada que comença un nou episodi.	Assigna el valor de la massa de l'agent1 en l'entrenament amb currículum learning. Només a l'agent1 perquè en fer-ho a tots dos s'anul·laria l'efecte.
CollectObservations (VectorSensor sensor)	És el mètode que farceix el vector d'observacions que a cada iteració farà servir l'agent per a prendre una decisió.	Recollim les observacions descrites a l'apartat anterior i les annexem al vector d'observacions.
OnActionReceived (float[] vectorAction)	Funció que rep el vector d'accions generat pel model de decisió de l'agent, les quals hem d'interpretar i aplicar.	Traduïm el valor de l'acció rebuda al moviment corresponent de l'agent. Apliquem una petita recompensa negativa amb el mètode *AddReward*(-1/5000) i finalitzem l'episodi amb *EndEpisode*().
Heuristic()	Permet farcir el vector d'accions de manera programada o mitjançant entrades del controlador que fa servir un usuari.	En base a les entrades de l'usuari que hem vist a l'apartat 2.2.1, indiquem el valor corresponent al vector d'accions.

Taula 10.Funcions de l'agent.

A banda d'aquests nous mètodes, en *FixedUpdate()* afegim el control de superació dels límits de l'escenari, en cas de succeir apliquem la recompensa negativa amb la funció A*ddReward(-1)*, finalitzem l'episodi amb *EndEpisode()* i cridem a la funció Place*Agent()* de l'àrea. Aquí mateix també afegim la lectura del color de la bandera que ens permet determinar la observació de l'estat de la bandera.

Disposem del codi d'aquesta versió de l'agent a l'annex: AgentController_v1.cs

3.5.3 Bandera

A l'algoritme de la bandera que hem vist a la secció 2.2.2 tant sols hi hem d'afegir l'aplicació de les recompenses i la finalització de l'episodi quan un agent ha completat la captura de la bandera.

Per tant, quan captura l'agent1, afegim:

agent1.AddReward(+1);
agent2.AddReward(-1);
agent1.EndEpisode();
agent2.EndEpisode();

I a la inversa per a la captura de l'agent2.

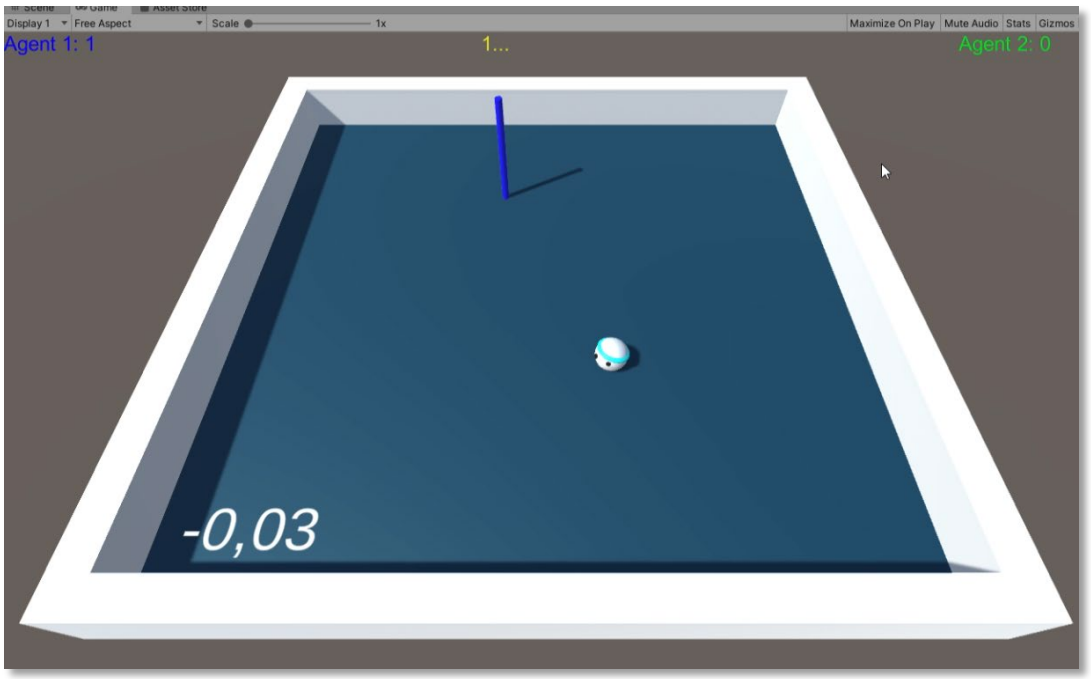
Disposem del codi d'aquesta versió de la bandera a l'annex: Flag.cs

3.6 Execució de l'entrenament

En aquest apartat repassem tots els entrenaments que s'han anat realitzant durant el desenvolupament de l'escenari inicial d'aprenentatge. La versió descrita en els punts anteriors és la obtinguda després d'aplicar tots els ajustos que hem considerats necessaris durant el procés d'aprenentatge.

3.6.1 Un sol agent captura la bandera

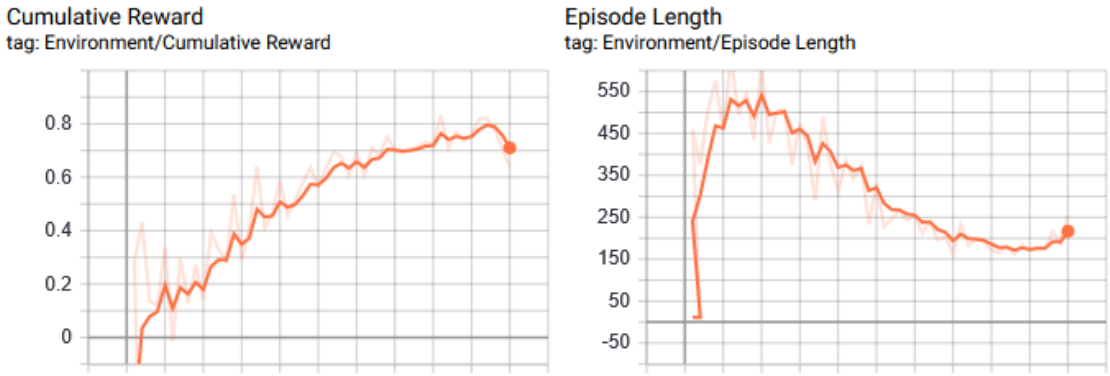
El primer entrenament que realitzem és el d'un sol agent capturant la bandera, sense rival i sense currículum learning. Pretenem testejar el sistema de la manera més simple possible i verificar que funciona correctament.



Il·lustració 15. Escenari d'entrenament amb un sol agent.

En aquesta primera versió de l'entorn apliquem una recompensa gradual per la captura de la bandera, on a cada segon de captura l'agent obté una recompensa de +0,25. Tampoc finalitzem els episodis en completar la captura.

Els resultats són els següents:

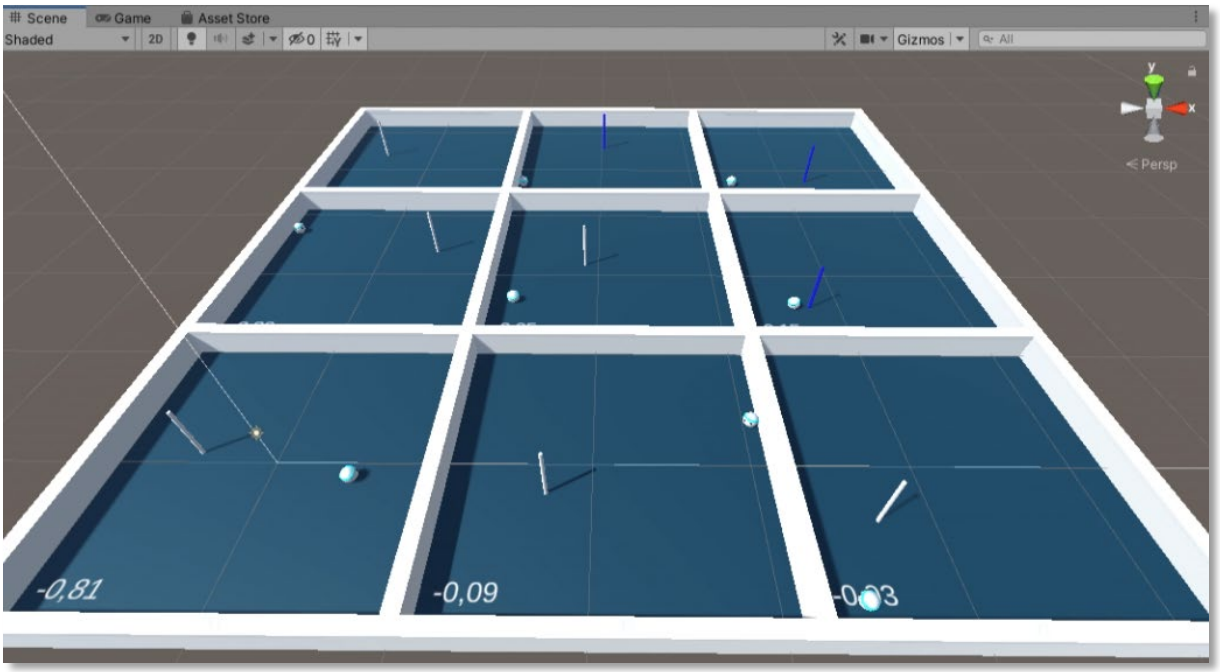


Il·lustració 16. Resultats del primer entrenament amb un sol agent.

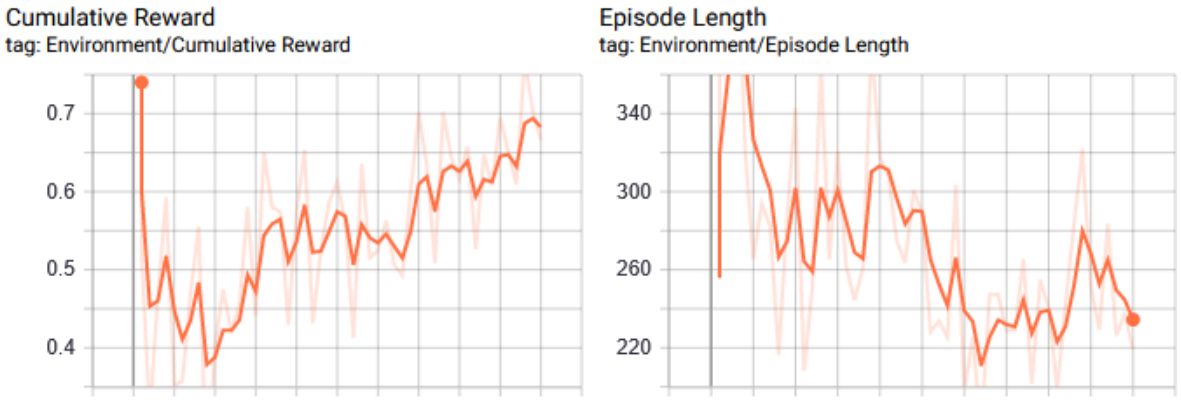
Una primera conclusió que podem treure és que l'entrenament és efectiu, ja que l'agent localitza la bandera i va a capturar-la cada vegada en menys temps, però no sembla òptim perquè no arriba a la màxima recompensa i de fet es produeix un efecte de sobre-entrenament cap als 500k episodis, on la recompensa acumulada comença a descendir. Això pot ser degut al mecanisme de recompensa gradual, quan l'agent obté recompenses mentre està capturant sense fer cap acció directa (només esperar).

3.6.2 Paral·lelització dels entrenaments

El segon entrenament consisteix en replicar l'àrea 9 vegades en el mateix escenari, de tal manera que l'entrenament sigui de 9 agents en paral·lel alimentant el mateix model.



Il·lustració 17. Escenari d'entrenament amb paral·lelització. Un sol agent.



Il·lustració 18. Resultats entrenament un sol agent amb paral·lelització.

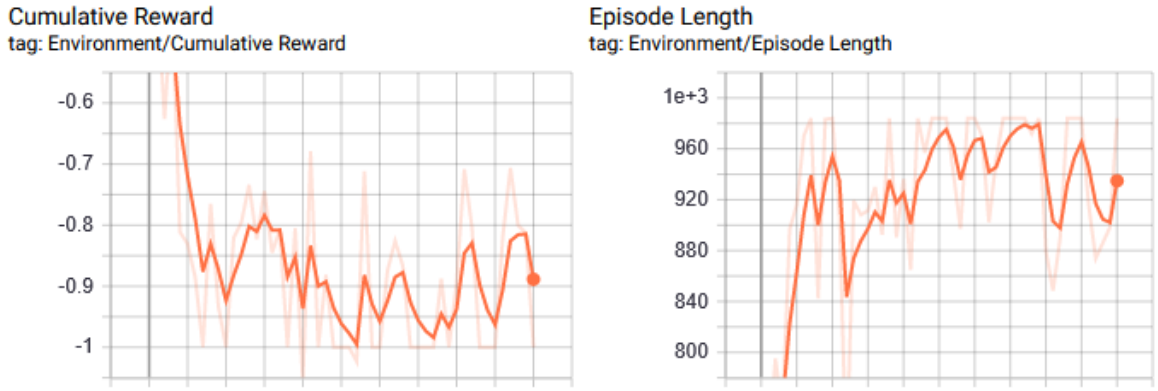
Els resultats mostren un patró molt similar al cas anterior, però amb uns màxims i mínims molt accentuats, generats per l'aleatorietat dels diferents agents entrenant simultàniament.

Podem veure una acceleració en la recompensa acumulada (des del principi ens situem en un valor superior al 0,4 quan en el cas anterior no s'hi arriba fins als 150k episodis), així com en el descens de la durada dels episodis. Per tant es corrobora que la paral·lelització permet accelerar els resultats dels entrenaments notablement sense incrementar el nombre d'episodis.

3.6.3 Eliminació de la recompensa gradual durant la captura

Arran dels resultats obtinguts en els anteriors entrenaments veiem necessari eliminar el comportament gradual de la recompensa durant la captura, i passem a tenir una única recompensa positiva de +1 en completar la totalitat de la captura. Mantenim l'entrenament paral·lel de 9 àrees.

Executem i visualitzem els resultats. Seguim amb un màxim de 500k episodis:

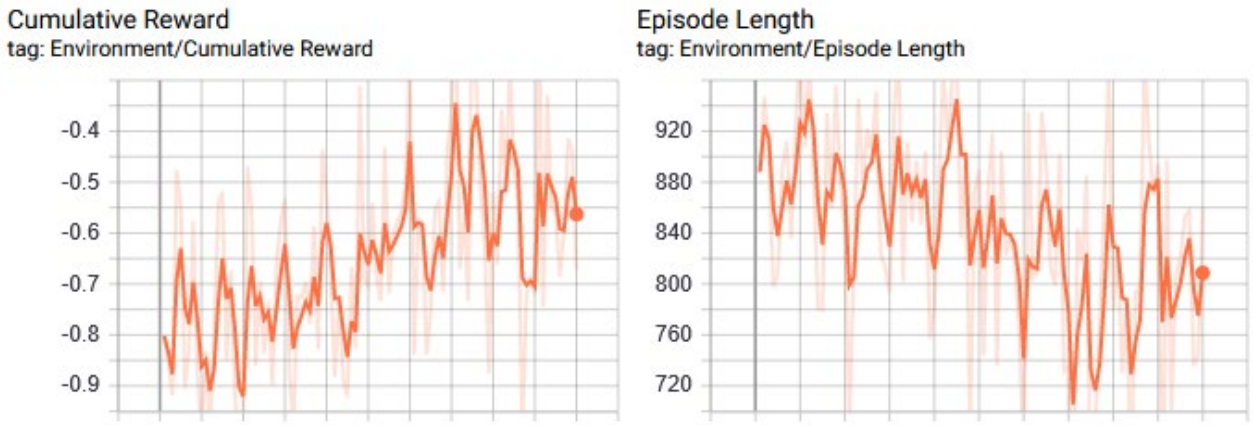


Il·lustració 19. Resultats entrenament un sol agent sense recompensa gradual durant la captura.

Clarament podem veure que el model ha empitjorat força, sent incapaç d'assolir una recompensa acumulada positiva ni superior al -0,9. Com que durant els 3 segons de captura l'agent està rebent una recompensa negativa per cada acció presa, i no pot realitzar cap acció per obtenir una recompensa positiva, no correlaciona correctament el fet de capturar la bandera amb la recompensa rebuda al cap de 3 segons.

Arrel de la darrera execució sembla clar que no podem recompensar a l'agent mentre espera la captura de la bandera, ni tampoc penalitzar-lo, per tant provem de no aplicar cap tipus de recompensa en aquest interval.

Executem incrementant a 1M el nombre d'episodis per arribar a un punt més estable:



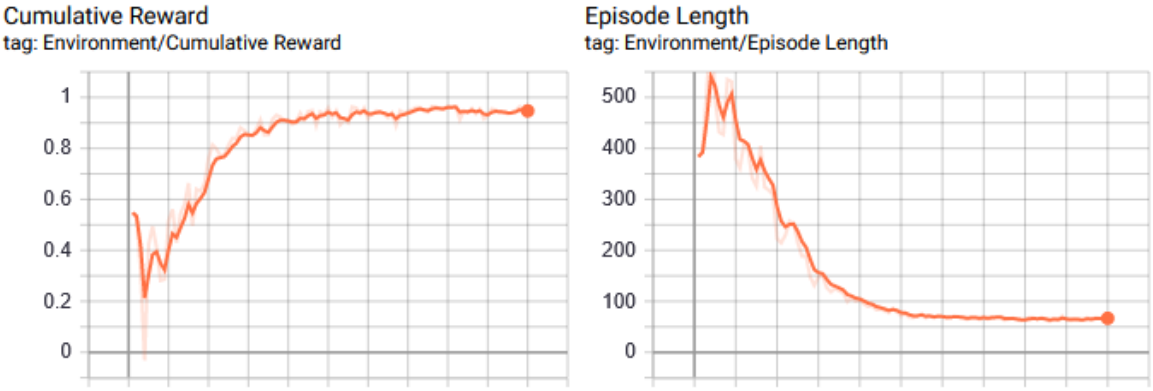
Il·lustració 20. Resultats entrenament un sol agent sense recompensa gradual ni negativa durant la captura.

Els resultats indiquen una lleugera millora i un aprenentatge molt lent, ja que després de 1 milió d'episodis en poc casos supera una recompensa acumulada de -0,4.

3.6.4 Aplicació de curriculum learning

Per intentar facilitar i accelerar l'aprenentatge de l'agent, procedim a aplicar curriculum learning (CL) tal i com hem descrit durant el disseny, però donat que seguim amb un sol agent, només variarem l'atribut de distància màxima de la bandera.

Executem entrenament amb un màxim d'1M d'episodis:



Il·lustració 21. Resultats entrenament un sol agent amb curriculum learning.

Els resultats són molt positius, sobre els 400k episodis arribem al 90% de la recompensa màxima i amb 250k ja superem qualsevol resultat màxim anterior. La durada mínima de l'episodi s'estabilitza a 70 sobre els 500k episodis. El fet de començar amb la bandera en una posició concreta permet al model correlacionar el comportament de la bandera amb la recompensa, i un cop assolit això, podem anar ampliant la distància màxima d'aparició de la bandera sense penalitzar l'entrenament.

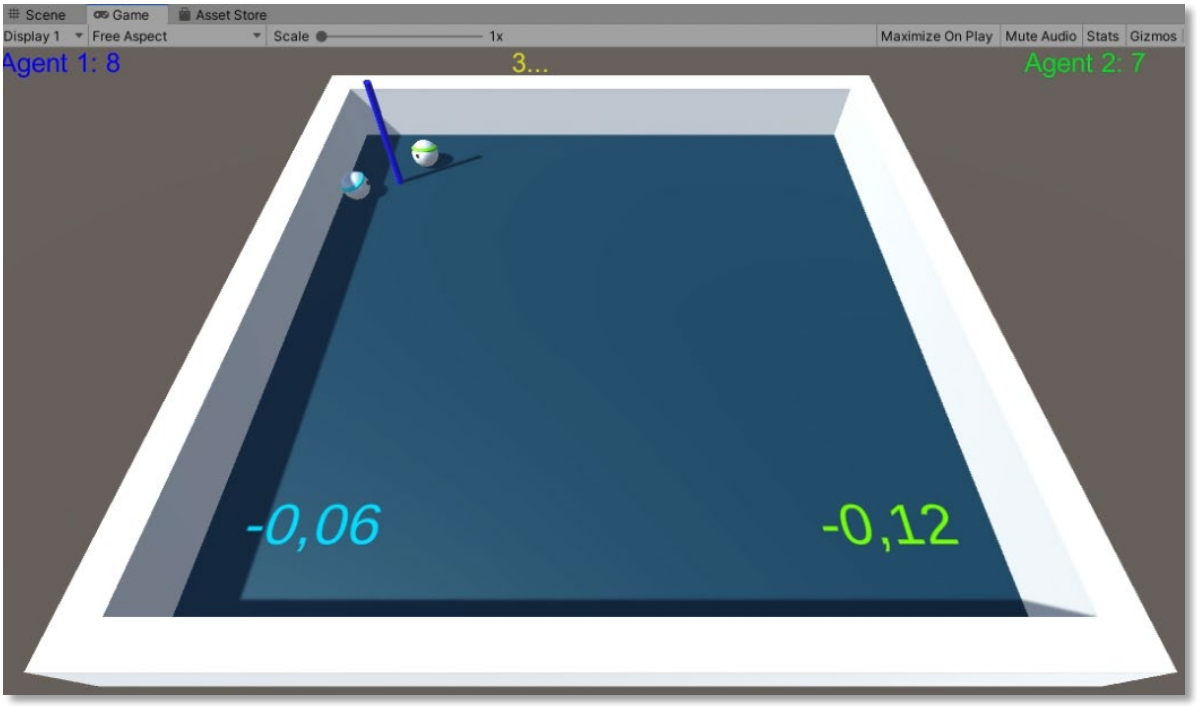
Al model obtingut l'anomenem **IAv0**.

3.6.5 Incorporació del segon agent

Un cop disposem del model entrenat per a realitzar la tasca, l'apliquem sobre els dos agents i en executar la simulació el resultat és el que ens imaginem: cada agent intenta completar la tasca de capturar la bandera sense tenir present a l'altre.

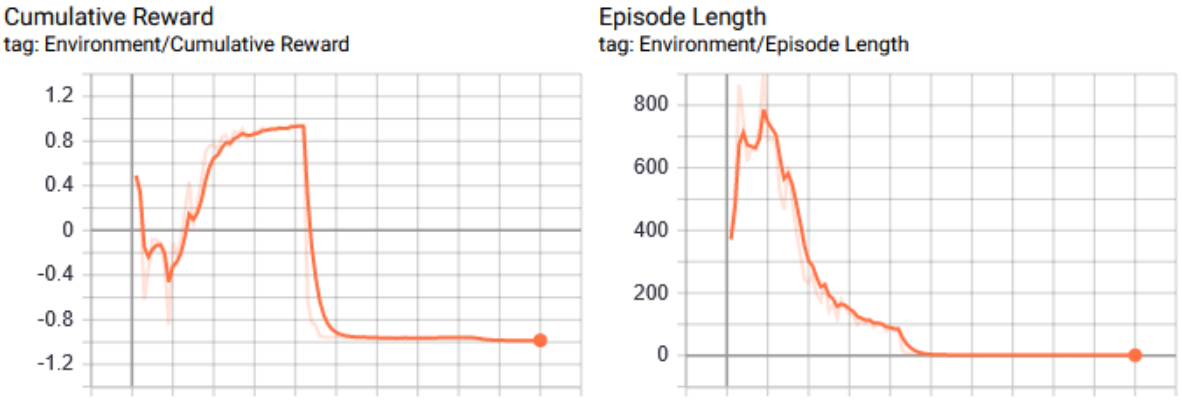
Per tant, per obtenir un comportament competitiu entre els agents, haurem d'executar entrenaments on estiguin els dos agents en l'escenari, com és obvi. Aquests entrenaments els farem de manera progressiva, entrenant a únicament un dels dos agents mentre que l'altre farà servir per inferència un model ja entrenat prèviament.

En el primer experiment entrenarem a l'agent1 i inferim a l'agent2 el model IAv0 obtingut en les passes anteriors.



Il·lustració 22. Escenari d'entrenament amb dos agents.

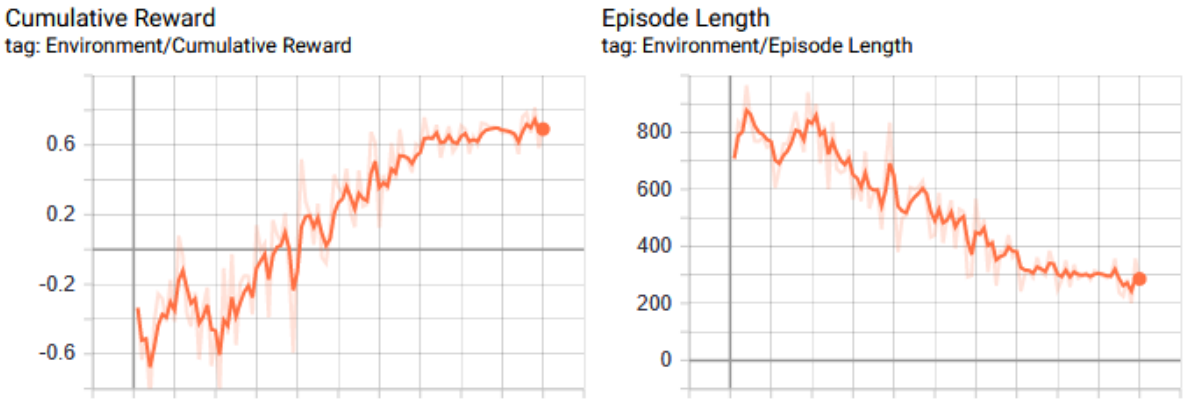
Executem entrenament d'1M d'episodis:



Il·lustració 23. Resultats primer entrenament contra un agent IAv0.

Els resultats obtinguts són força estranys, doncs sembla que la recompensa acumulada i la durada dels episodis evolucionen positivament, fins a obtenir una baixada dràstica de les dues mesures. Observem la simulació i detectem un error: quan per efecte d'una col·lisió un dels agents surt de l'escenari i cau al buit, la posició de la bandera és reinicialitza, però no la de l'agent. De tal manera que l'agent cau indefinidament i la bandera es mou de posició de manera constant i erràtica, impossibilitant ser capturada.

Corregim el defecte fent que la posició de l'agent també es reinicialitzi en sortir dels límits de la superfície i tornem a executar 1M episodis:



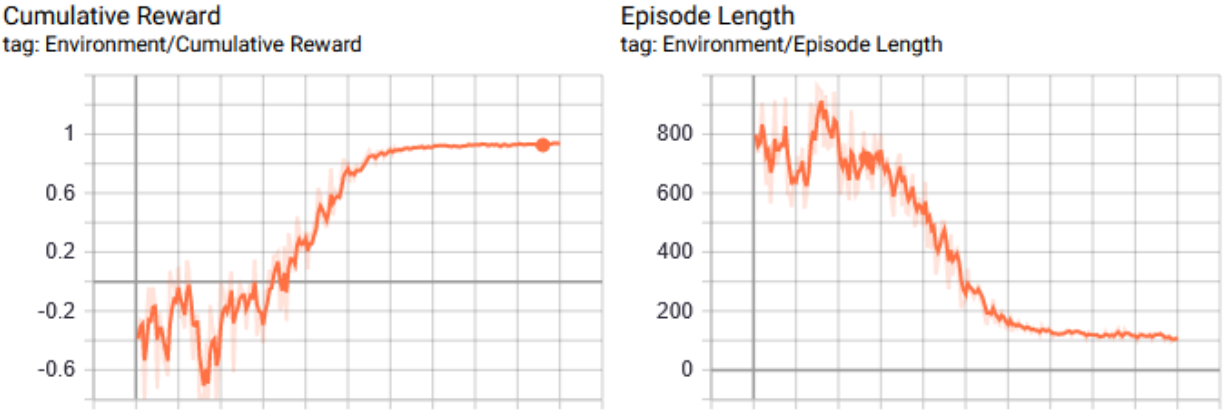
Il·lustració 24.Resultats segon entrenament contra un agent IAv0.

L'agent que s'està entrenant no és capaç d'arribar a guanyar al ja entrenat en la totalitat dels episodis, arribant a una recompensa acumulada de 0,7 al milió d'episodis.

3.6.6 Optimitzacions dels algoritmes

Donat que en el darrer entrenament no arribem als resultats desitjats, efectuem algunes optimitzacions del codi com ara, en la observació de la velocitat de l'agent no guardar la corresponent a l'eix y (ja que el moviment només es sobre els eixos x i z), i aplicar el control dels límits de l'escenari també en els eixos x i z (fins al moment s'aplicava només a l'eix y, quan l'agent cau al buit, però en l'entrenament amb 9 àrees paral·leles l'agent pot anar a parar a una àrea anexa i afectar negativament a la tasca dels "veïns").

També, pel fet d'estar augmentant la complexitat de la tasca, necessitarem més episodis així que ampliem l'entrenament a 2M episodis:



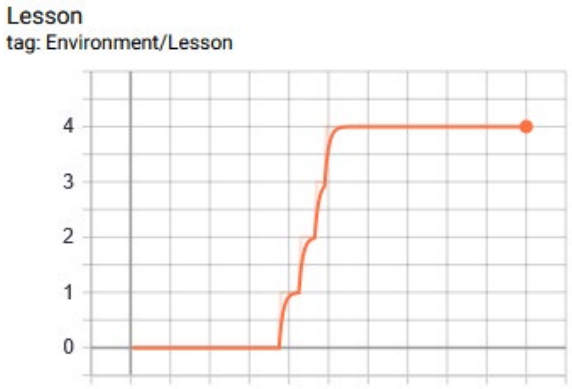
Il·lustració 25. Resultats tercer entrenament contra un agent IAv0.

Els resultats són favorables, l'agent aconsegueix al voltant del 1,2M episodis assolir una recompensa acumulada mitja del 90% i per tant guanyar a l'altre agent pràcticament en la totalitat dels casos.

També podem observar que l'agent ha desenvolupat les següents habilitats per tal de guanyar al seu rival:
- Xocar repetidament contra la bandera fins que la captura.
- Interposar-se entre la bandera i l'agent contrari quan està en procés de captura.

A aquesta nova versió del model la anomenem **IAv1**.

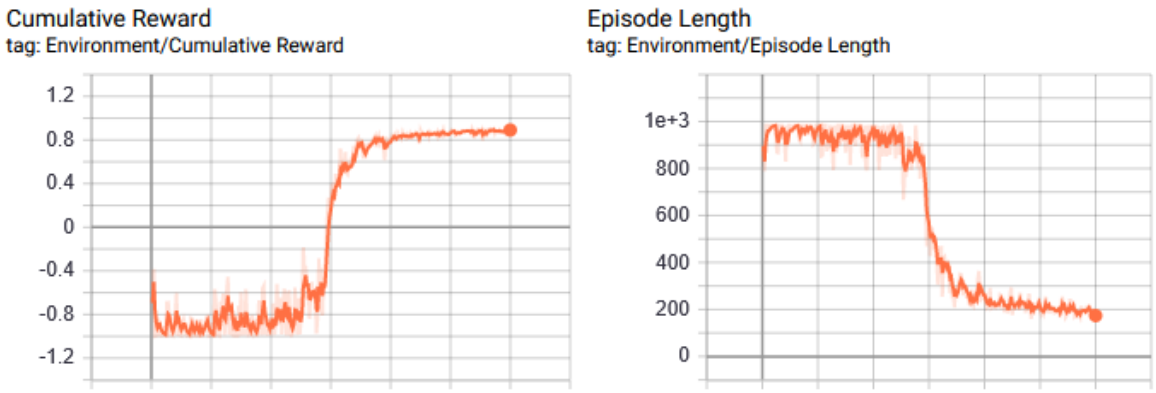
Aprofitem per mostrar un gràfic relacionat amb el currículum learning, on podem veure el salt d'una lliçó a la següent, és a dir la superació dels límits de recompensa acumulada que modifica la variable de distància màxima de la bandera:



Il·lustració 26. Gràfic curriculum learning.

3.6.7 Segona volta. Entrenament contra la nova versió del model

Per seguir millorant el model, inferim a l'agent2 el resultant dels darrers entrenaments, IAv1, i entrenem novament a l'agent1, aquesta vegada fins a 3M d'episodis:



Il·lustració 27. Resultats entrenament contra un agent IAv1.

Observem que durant la meitat de l'entrenament (un milió i mig d'episodis) l'agent1 és incapaç de vèncer a l'agent2, però en aquest moment hi ha un punt d'inflexió i el comença a guanyar, arribant a una recompensa acumulada mitja del 90% cap als 3M episodis.

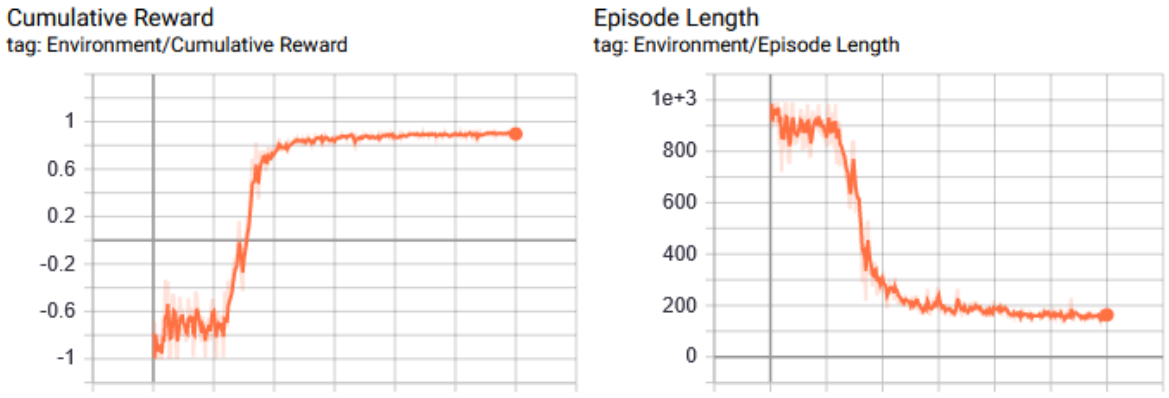
Les millores que podem observar visualment d'aquesta versió del model, és que l'agent localitza i es dirigeix de manera més ràpida cap a la bandera, i "insisteix" més a capturar-la, però no observem cap nova habilitat.

3.6.8 Tercera volta. Ampliació del curriculum learning

Amb la finalitat d'intentar desencadenar l'aprenentatge d'alguna habilitat nova tornem a fer la operació anterior i inferim a l'agent2 el darrer model generat, IAv2, i entrenem a l'agent1 per tal que intenti superar-lo.

També sembla necessari donar certa avantatge a l'agent que s'entrena respecte l'agent pre-entrenat si volem que faci alguna cosa diferent. En aquest sentit, ampliem el curriculum learning existent on variem la distància màxima d'aparició de la bandera, afegint una nova variant: la massa de l'agent que s'entrena. Començarem donant-li el doble de massa que el seu oponent i l'anirem reduint progressivament fins a tenir la mateixa.

Executem un entrenament de 3M d'episodis:



Il·lustració 28. Resultats entrenament contra un agent IAv1 ampliant el curriculum learning.

Els resultats són molt positius i l'agent1 comença a guanyar a l'agent2 molt abans que en l'experiment anterior, sobre els 750K episodis, i s'estabilitza en un 90% de la recompensa acumulada mitja cap als 2M episodis.

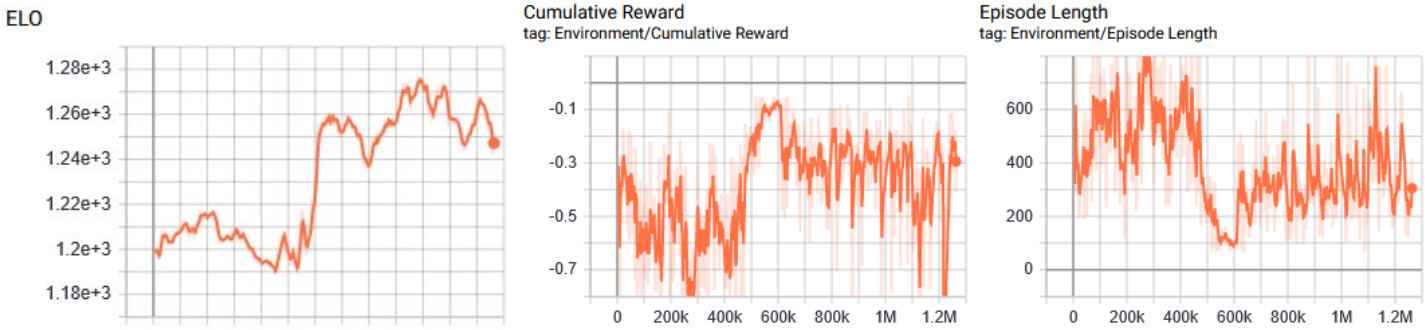
A més, en aquest cas observem canvis en el comportament de l'agent entrenat:
- Empenya a l'agent rival més sovint per apartar-lo de la bandera.
- Sembla que calcula el moment precís per robar la captura a l'agent rival quan aquest s'està allunyant d'ella.

A aquesta nova versió del model l'anomenem **IAv2**.

3.6.9 Selfplay

Finalment, provarem l'entrenament amb la configuració *selfplay* que hem descrit en el capítol 3.4.5, on l'agent1 s'entrenarà simultàniament contra l'agent2 que anirà alternant el model inferit cada 3000 episodis.

Executem 1.2M episodis (la intenció era executar 3M però pel algun motiu se'ns atura als 1.2). Tot i així, obtenim resultats:

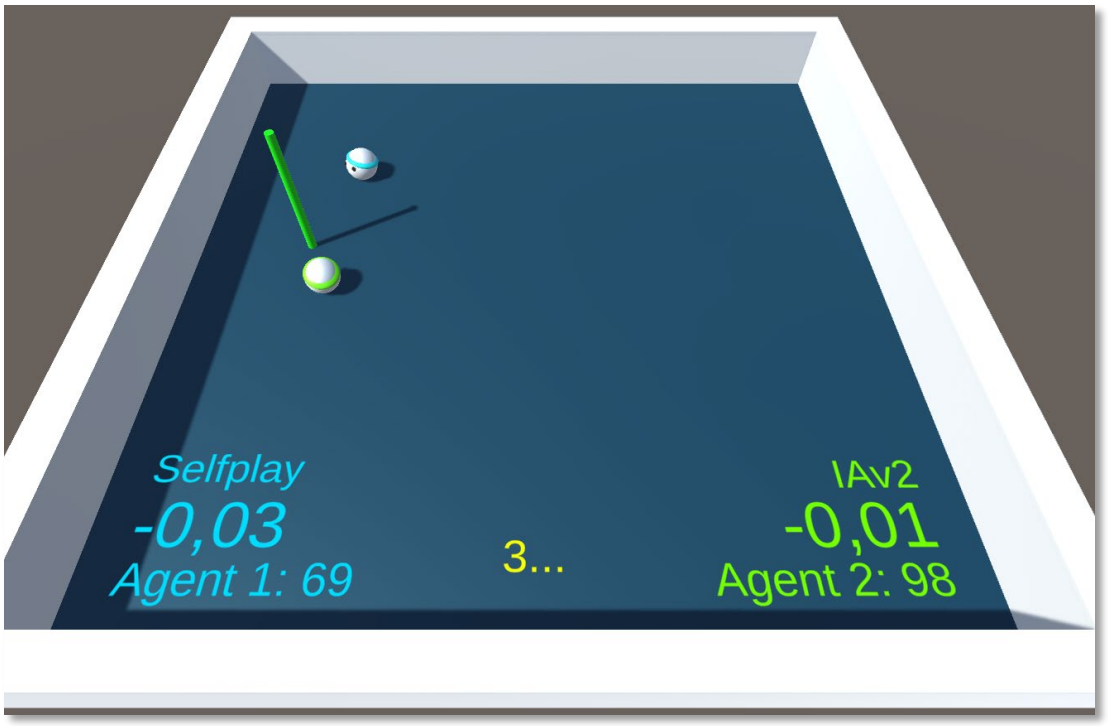


Il·lustració 29. Resultats entrenament selfplay.

En aquest cas tenim la mesura de la puntuació ELO, que ens permet avaluar la probabilitat de que l'agent1 capturi la bandera en comptes de l'agent2, i veiem que té un creixement ascendent. Aquesta nova mesura que ens apareix amb l'entrenament *selfplay* és degut a que ja no té sentit observar les variables que estàvem mesurant. Veiem que la recompensa acumulada sembla que no és un factor indicatiu de la millora del model entrenat, perquè al anar variant el model que s'infereix a l'agent2, aquest sempre presenta una dificultat superior per a l'agent1, però això no implica que el model entrenat no millori. Per aquest motiu és millor observar l'evolució del ELO. Quant a la longitud dels episodis, passa exactament el mateix.

Per poder contrastar el model entrenat amb Selfplay amb el model entrenat realitzant 3 iteracions manuals, IAv2, inferim un dels models a cada agent i executem la simulació per visualitzar el comportament.

Podem veure com el model entrenat amb Selfplay, tot i haver estat entrenat amb aproximadament un 20% del total dels episodis que s'han aplicat en el conjunt de les iteracions per obtenir IAv2, és capaç de capturar la bandera en dues de cada cinc jugades. Per tant podem corroborar que la política selfplay és més eficient i idònia per el tipus de simulació competitiva que hem dissenyat.



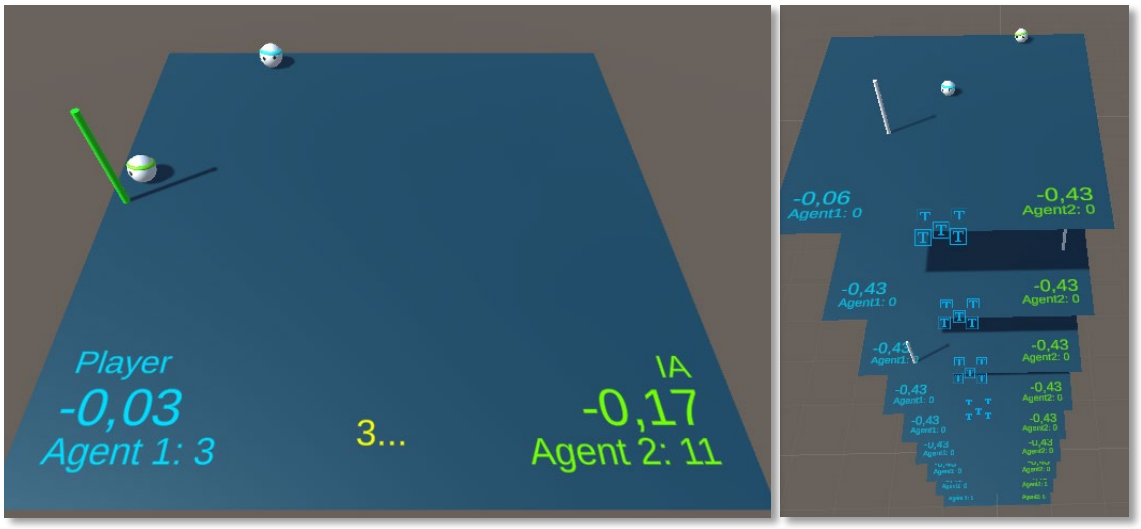
Il·lustració 30. Demostració de joc Selfplay vs IAv2.

4. Iteracions de l'entorn d'aprenentatge

En el present apartat expliquem el detall de les 3 ampliacions de l'entorn dutes a terme i els resultats de l'aplicació de l'aprenentatge per reforç.

4.1 Sense murs perimetrals

En el primer cas, retirem les parets que limiten l'espai de joc i que eviten que els agents es precipitin cap al buit.

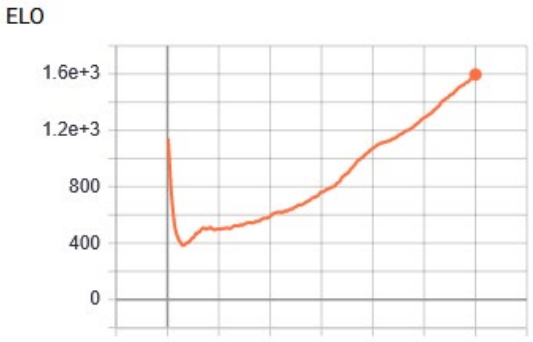


Il·lustració 31. Escenari sense murs perimetrals.

A nivell funcional dels agents i de l'algoritme d'aprenentatge no necessitem fer cap variació, ja que els agents ja tenen establerta una recompensa negativa màxima en cas de sortir de la superfície de joc.

Tant sols hem d'ajustar la disposició de les 9 àrees en paral·lel perquè disposades en forma de quadrícula, sense els murs, els agents poden arribar a una àrea veïna i alterar el seu entrenament. Així doncs, tal i com es veu a la imatge anterior, disposem les àrees en forma de torre perquè en cas de caiguda no es puguin afectar entre elles.

Duem a terme l'execució de l'entrenament amb 3M d'episodis i observem com hi ha un progressió positiva de l'ELO.



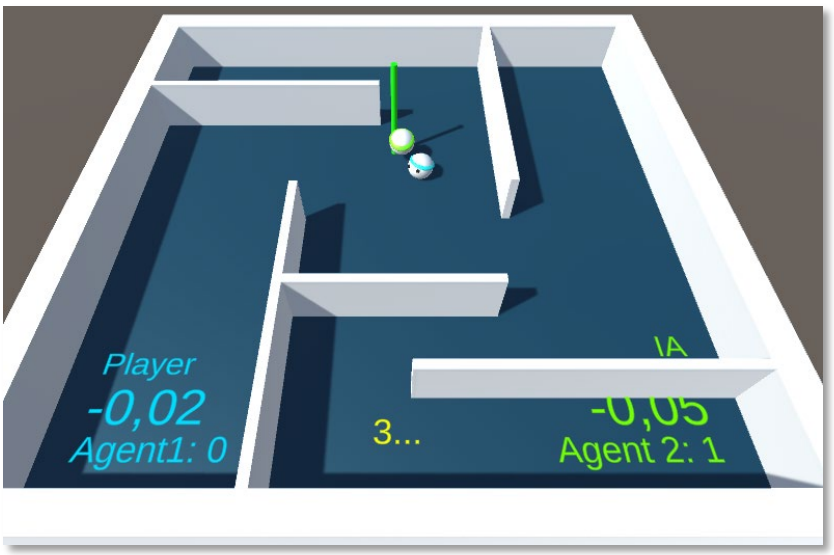
Il·lustració 32. Resultats entrenament sense murs perimetrals.

A continuació inferim el model generat a un dels agents i executem una simulació per veure si ha adoptat algun comportament destacable. Observem el següent:

- L'agent es mou de manera més lenta i precisa, per evitar caure.
- En cas d'acostar-se l'agent contrari, l'empenya fins a fer-lo caure (de manera poc polida, doncs gairebé sempre acaben precipitant-se tots dos).

4.2 Murs obstaculitzants

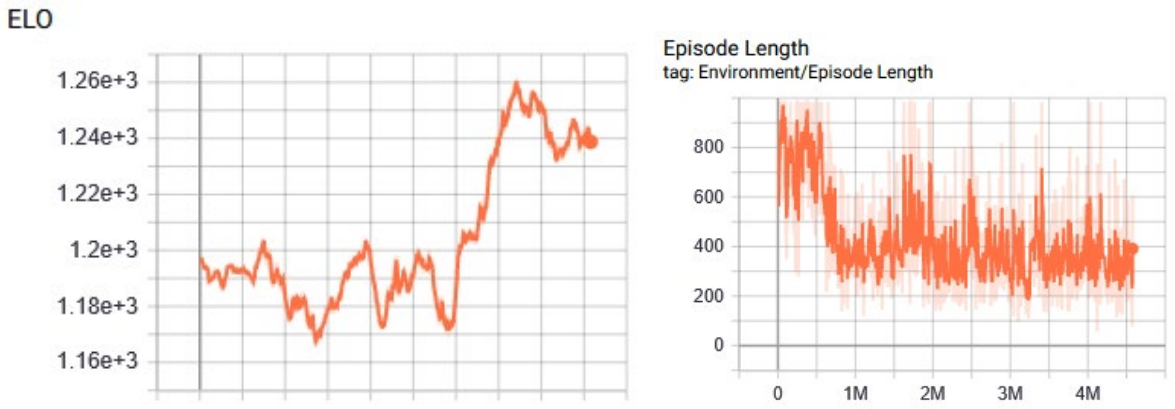
A la segona iteració de l'entorn afegim una sèrie de murs sobre l'escenari, que actuen com a obstacles i generen espais estancs (habitacions), amb la finalitat de provar la tècnica *curiosity*, que potencia el comportament exploratori dels agents.



Il·lustració 33. Escenari amb murs obstaculitzants.

A nivell d'algoritmítica implementem un *trigger* en la bandera que fa que en cas de trobar-se a la mateixa posició que un mur interior, es torni a moure a una posició aleatòria, evitant sol·lapaments.

En aquest cas necessitem executar un entrenament de 5M d'episodis per començar a obtenir resultats favorables, on l'ELO comença a ascendir. També veiem que, malgrat això, la llargada dels episodis es manté elevada.

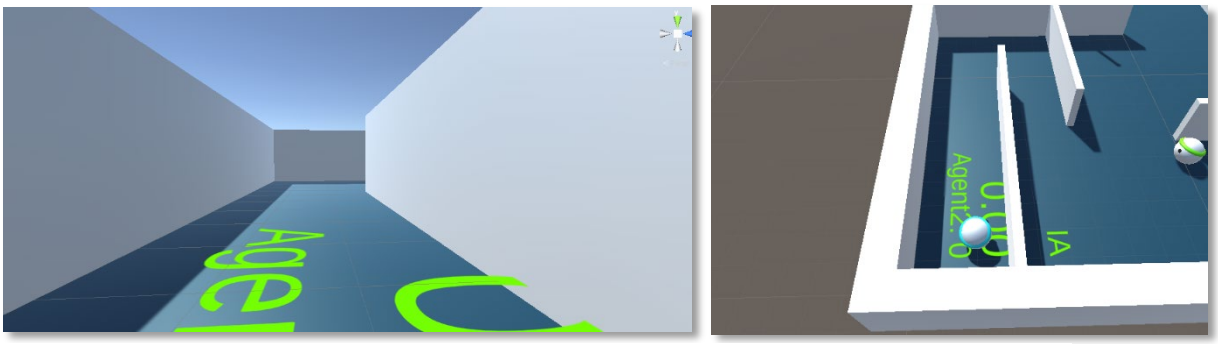


Il·lustració 34. Resultats entrenament amb murs obstaculitzants.

Com sempre, apliquem el model generat a un agent i observem com es comporta en una simulació. Veiem que:

- Aprofita un defecte de l'escenari que provoca que pugui atravessar els murs si hi col·lisiona amb una inèrcia suficient.
- Es mou de manera exploratòria, desplaçant-se a altres espais si es troba encallat en un racó. El paràmetre *curiosity* resulta efectiu.
- Si la bandera apareix en un racó prou amagat, li resulta molt difícil localitzar-la i triga força estona en arribar a capturar-la.

Per aconseguir millorar el temps de localització i captura de la bandera, una bona pràctica és posar-se al nivell de les observacions de l'agent per veure com es podrien millorar:



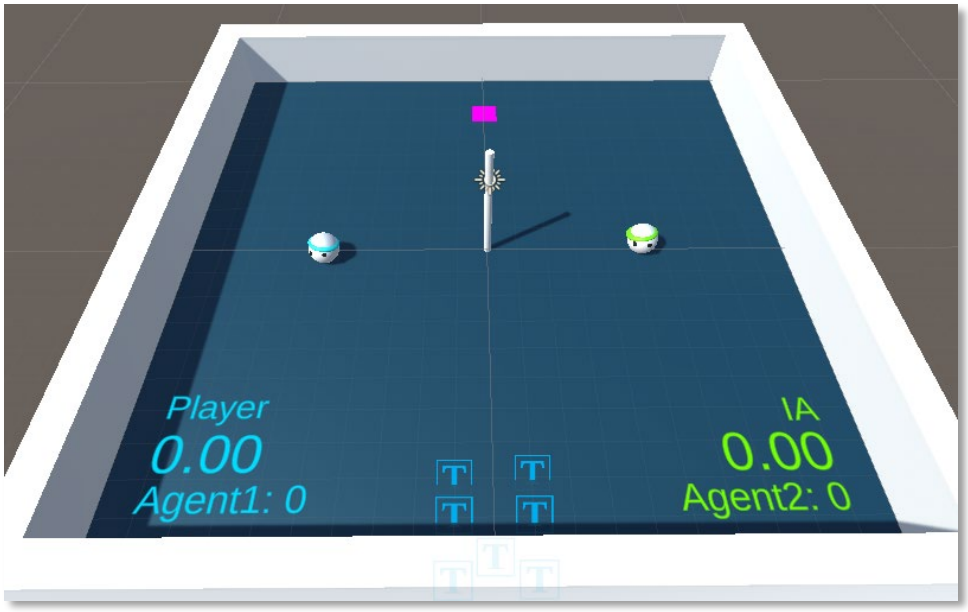
Il·lustració 35. Punt de vista de l'agent en una posició complexa.

Tal com podem veure, entre els murs és molt difícil localitzar la bandera i per tant només identifiquem dues possibles solucions:

- Seguir incrementant el nombre d'episodis fins a perfeccionar el mecanisme exploratori.
- Afegir una nova observació de l'escenari (com ara el mòdul de la distància entre l'agent i la bandera, encara que no la tingui en el seu rang de visió).

4.3 Botó d'avantatge

En aquesta tercera i darrera volta de modificacions de l'entorn d'aprenentatge, hi pretenem afegir un nou objecte amb què interactuar, aportant dinamisme i més varietat de decisions als agents.



Il·lustració 36. Escenari amb interruptor d'avantatge.

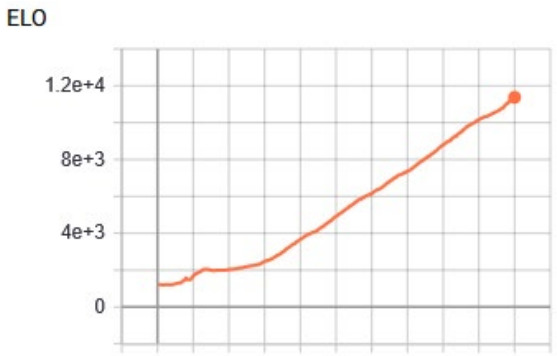
Incorporem l'objecte botó a l'escenari (rajola de color lila) i en definim el comportament:

L'agent, en cas de col·lisionar amb el botó, si aquest està actiu, obté un increment de massa i de velocitat durant 10 segons. El botó passa a estar inactiu durant 15 segons. Quan finalitzen els temps pertinents l'agent torna a tenir la massa i velocitat inicials, i el botó es reactiva.

Per tant, necessitem modificar el codi de l'agent per implementar la interacció de col·lisió i la modificació de les seves propietats, el codi de l'àrea per gestionar el posicionament del botó a una nova ubicació aleatòria quan finalitza un episodi, i al codi de la bandera simplement necessitem reanomenar algunes classes per coherència amb la nova àrea.

Tots els scripts modificats els tenim disponibles a l'annex: *AreaButton.cs, AgentController_v2.cs, FlagButton.cs*

Un cop aplicades i testejades totes les modificacions, executem entrenaments, apliquem el model a un agent i realitzem la simulació per observar si hi ha canvis en el comportament. En aquesta ocasió, necessitem realitzar fins a 10M d'episodis, per arribar a visualitzar alguna novetat:



Il·lustració 37. Resultats entrenament amb botó d'avantatge.

El comportament que visualitzem és que molt ocasionalment els agents interactuen amb el botó, tant sols trepitjant-lo quan està força proper de la trajectòria cap a la bandera. Per tant, en certa manera correlacionen un cert avantatge però no el suficient com per desplaçar-s'hi expressament.

Modifiquem l'avantatge proporcionada pel botó per veure si propiciem que s'utilitzi més, realitzant els següents canvis:

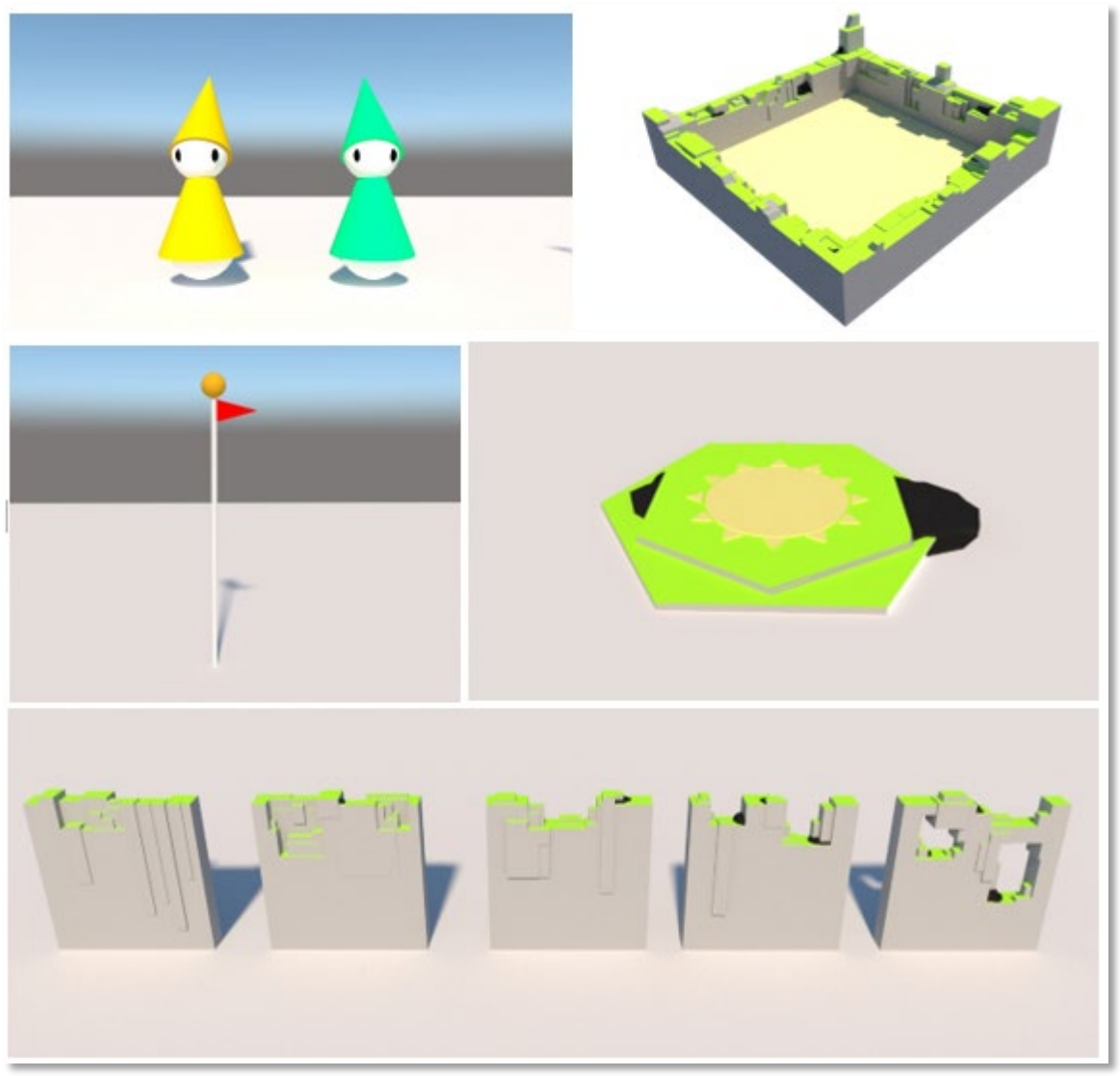
- En tocar el botó, l'agent contrari és desplaçat a la posició inicial.
- En tocar el botó, l'agent contrari queda immobilitzat durant 5 segons.

El resultat d'aquestes proves tampoc resulta positiu, els agents ignoren el botó. Segurament necessitaríem molts més episodis per tal que els agents arribin a correlacionar aquest avantatge, que no deixa de ser un factor que proporciona més probabilitats d'assolir l'objectiu (capturar la bandera) però d'una manera molt indirecta.

4.4 Aplicació de dissenys artístics

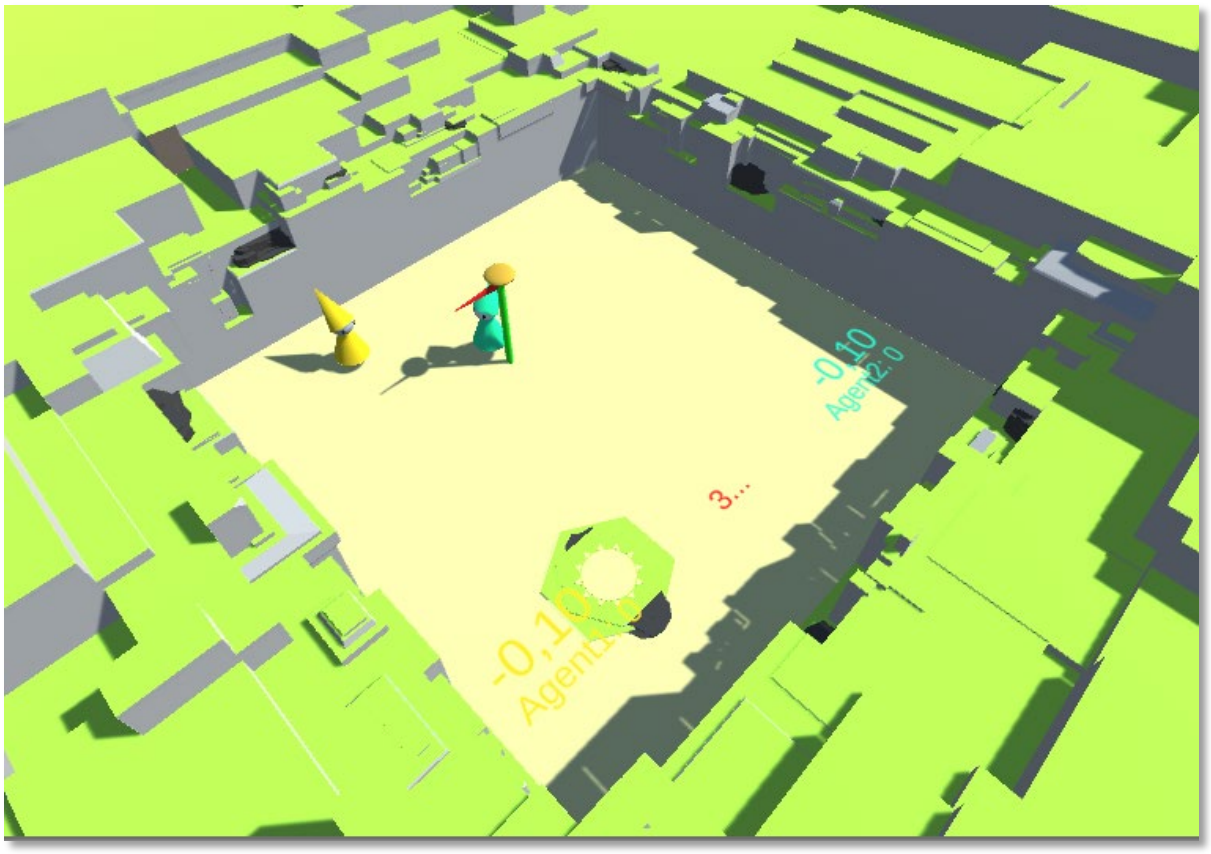
En aquest cas no estem parlant d'una iteració de l'entorn d'aprenentatge, però sí d'una millora de l'entorn experimental que hem desenvolupat per mitjà de l'aplicació de dissenys artístics.

Els dissenys que mostrem a continuació han estat cedits per una persona propera que ha volgut col·laborar en el treball de manera desinteressada:



Il·lustració 38. Disseny artístic dels agents, escenari, bandera, botó i murs.

La tasca en aquest cas ha estat traslladar aquests dissenys al nostre entorn, aplicar-los i verificar el seu correcte encaix sense afectar al funcionament del sistema. El resultat és el següent:



Il·lustració 39. Entorn amb dissenys artístics aplicats.

4.5 Versió interactiva en format web

Com a darrera tasca del projecte, pretenem posar a disposició de l'equip avaluador del treball una versió interactiva de l'entorn virtual amb els diferents escenaris desenvolupats i amb els diferents models neuronals entrenats aplicats sobre els agents. Aquesta versió ha de ser fàcilment accessible i executable, per tant la solució òptima és el web.

Per dur-ho a terme aprofitem la característica de Unity que permet realitzar una compilació del projecte en format *HTML5.*

També necessitem desenvolupar una interfície de menú a través del qual es puguin accedir als diferents escenaris de demostració, que implementem amb un element *Canvas.* Hi disposem una sèrie de botons, cadascun per accedir a una demostració, que utilitzant la funció *SceneManager.LoadScene(escena)* ens carreguen l'escenari corresponent. S'adjunta aquest codi com a annex: *MainMenu.cs*.

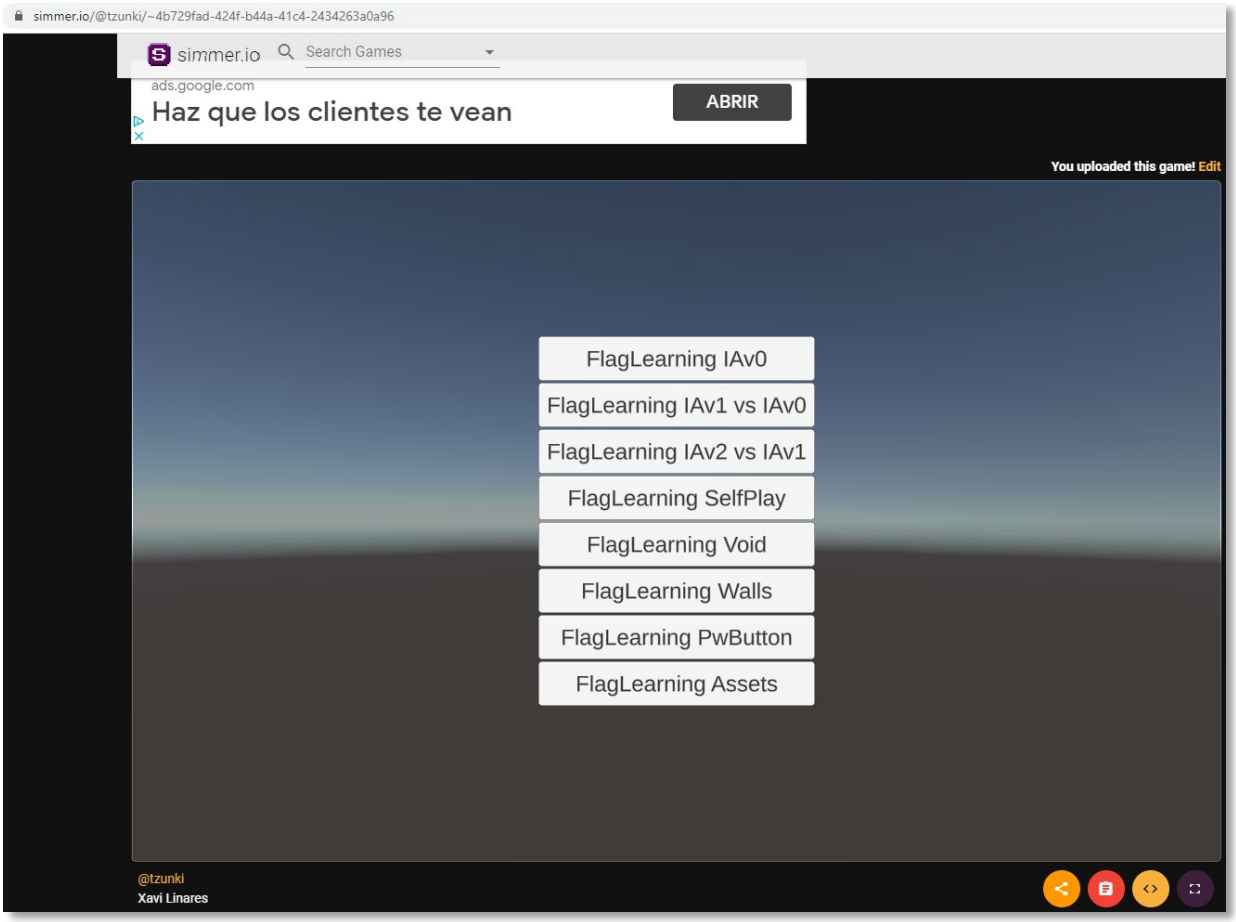
La relació de demostracions segons el literal indicat a cada botó és la següent:

Botó	Escenari demostració
FlagLearning IAv0	Escenari inicial de captura de bandera. Agent1 (blau) és controlat pel jugador (heurístic). Agent 2 (verd) infereix el model IAv0.
FlagLearning IAv1 vs IAv0	Escenari inicial de captura de bandera. Agent1 (blau) infereix el model IAv1. Agent 2 (verd) infereix el model IAv0.
FlagLearning IAv2 vs IAv1	Escenari inicial de captura de bandera. Agent1 (blau) infereix el model IAv1. Agent 2 (verd) infereix el model IAv0.
FlagLearning Selfplay	Escenari inicial de captura de bandera. Agent1 (blau) infereix el model Selfplay. Agent 2 (verd) infereix el model IAv2.
FlagLearning Void	Escenari sense murs perimetrals. Agent1 (blau) és controlat pel jugador (heurístic). Agent 2 (verd) infereix el model entrenat en l'escenari.
FlagLearning Walls	Escenari amb murs obstaculitzants. Agent1 (blau) és controlat pel jugador (heurístic). Agent 2 (verd) infereix el model entrenat en l'escenari.
FlagLearning PwButton	Escenari amb botó d'avantatge. Agent1 (blau) és controlat pel jugador (heurístic). Agent 2 (verd) infereix el model entrenat en l'escenari.
FlagLearning Assets	Escenari amb botó d'avantatge amb dissenys artístics. Agent1 (blau) és controlat pel jugador (heurístic). Agent 2 (verd) infereix el model entrenat en l'escenari amb botó d'avantatge.

Taula 11. Relació de botons i escenaris de demostració.

Finalment, la plataforma que ens permet allotjar el projecte de manera accessible a través d'Internet és *Simmer.io*[9]. Creem un compte d'usuari, pugem els fitxers generats per Unity, i obtenim la següent URL:

https://simmer.io/@tzunki/~4b729fad-424f-b44a-41c4-2434263a0a96



Il·lustració 40. Versió interactiva en format web.

Cal tenir en compte que per a tornar al menú principal és necessari reiniciar l'aplicació. En aquest cas, és suficient amb recarregar la pàgina web mitjançant la funcionalitat pròpia del navegador utilitzat.

[9] simmer.io A place for Unity developers to share WebGL games [Data de consulta: 23/06/20] <https://simmer.io/>

5.Conclusions

Un cop finalitzat el treball, podem concloure que l'aprenentatge per reforç realment proporciona uns resultats molt positius quan es realitza en un entorn multiagent competitiu, perquè la constant evolució de l'agent contrari obliga a l'agent entrenat a optimitzar i depurar cada vegada més l'algoritme de decisions per superar-lo. Això, fins i tot porta a la detecció d'errors de desenvolupament del propi entorn d'aprenentatge, com per exemple poder travessar un mur que hauria de ser sòlid. Per tant, de l'experiència en podem extreure les següents aplicacions:

- Detecció de defectes en un entorn virtual.
- Optimització i automatització de tasques.

En quant a les dificultats trobades durant el treball, una d'elles rau en la falsa assumpció de que l'aprenentatge per reforç es pot aplicar de manera senzilla, segurament induïda pel concepte aprenentatge "automàtic", però és tot al contrari. Donat que en els entrenaments els agents realitzaran milions d'accions en l'entorn, qualsevol error o defecte salta de manera alarmant o afecta negativament als resultats. Dit d'una altra manera, la optimització de l'entorn i de l'algoritme d'aprenentatge ha de ser molt precisa, i hi cal invertir molt de temps.

L'altra dificultat té origen en el maquinari disponible per als entrenaments. Quan un entorn és prou complex, comença a requerir un volum elevat d'episodis d'entrenament per visualitzar resultats, i amb un ordinador personal això implica des de diverses hores d'execució fins a rangs inviables. Per tant, els resultats dels entrenaments no els tenim de manera gens immediata i això fa que qualsevol modificació sigui crítica.

Finalment s'ha arribat a la consecució de tots els objectius plantejats: disposem d'un entorn virtual, hi hem aplicat aprenentatge per reforç, i hi hem realitzat fins a 3 iteracions modificant l'escenari i aplicant entrenaments. Malgrat amb les dues iteracions finals no s'ha arribat a un entrenament òptim dels agents, sí hem visualitzat una evolució i canvis en el seu comportament. Considerem que la metodologia ha estat adequada.

Com a línia de treball futura, tenint en compte les limitacions introduïdes per el potencial de computació d'un ordinador, pensem que seria interessant integrar les comunicacions amb un sistema de computació distribuït on executar els entrenaments.

6. Glossari

Intel·ligència artificial (Artificial Intelligence, AI): camp de la ciència que té per finalitat la creació de sistemes que simulen processos d'intel·ligència

Aprenentatge màquina (Machine Learning, ML): branca de la intel·ligència artificial on es recerquen tècniques per simular el pensament humà, concretament els processos de presa de decisions i d'aprenentatge

Aprenentatge per reforç (Reinforcement Learning, RL): tècniques de l'àmbit de l'aprenentatge automàtic que consisteixen en un agent actuant sobre un entorn del qual en percep unes observacions i recompenses, i que té per finalitat maximitzar aquesta recompensa.

Aprenentatge per currículum (Curriculum Learning, CL): és una configuració d'aprenentatge per reforç que consisteix en incrementar gradualment la dificultat de la tasca a realitzar conforme l'agent va millorant en la recompensa obtinguda.

7. Bibliografia

Dot CSV [Data de visita 23/06/20] <https://www.youtube.com/channel/UCy5znSnfMsDwaLlROnZ7Qbg>

Emergent Tool Use from Multi-Agent Interaction [Data de visita 23/06/20] <https://openai.com/blog/emergent-tool-use/>

Unity Real-Time Development Platform [Data de visita 23/06/20] <https://unity.com/>

Roll-a-ball [Data de visita 23/06/20] <https://learn.unity.com/project/roll-a-ball-tutorial>

Unity-Technologies/ml-agents [Data de visita 23/06/20] <https://github.com/Unity-Technologies/ml-agents>

Reinforcement Learning Penguins (Part 1/4) | Unity ML-Agents [Data de visita 23/06/20] <https://www.immersivelimit.com/tutorials/reinforcement-learning-penguins-part-1-unity-ml-agents>

Reinforcement Learning Penguins (Part 2/4) | Unity ML-Agents [Data de visita 23/06/20] <https://www.immersivelimit.com/tutorials/reinforcement-learning-penguins-part-2-unity-ml-agents>

Reinforcement Learning Penguins (Part 3/4) | Unity ML-Agents [Data de visita 23/06/20] <https://www.immersivelimit.com/tutorials/reinforcement-learning-penguins-part-3-unity-ml-agents>

Reinforcement Learning Penguins (Part 4/4) | Unity ML-Agents [Data de visita 23/06/20] <https://www.immersivelimit.com/tutorials/reinforcement-learning-penguins-part-4-unity-ml-agents>

TensorFlow [Data de visita 23/06/20] <https://www.tensorflow.org/>

Artificial intelligence [Data de visita 23/06/20] <https://en.wikipedia.org/wiki/Artificial_intelligence>

Machine Learning [Data de visita 23/06/20] <https://en.wikipedia.org/wiki/Machine_learning>

Reinforcement Learning [Data de visita 23/06/20] <https://en.wikipedia.org/wiki/Reinforcement_learning>

simmer.io A place for Unity developers to share WebGL games [Data de consulta: 23/06/20] <https://simmer.io>

8. Annexos

8.1. AgentController_v1.cs

```
using UnityEngine;
using MLAgents;
using MLAgents.Sensors;

public class AgentController_v1 : Agent
{

      private Rigidbody rb;
      public float speed;
      private GameObject flag;
      private Area area;
      private Renderer flagRenderer;
      private Color flagColor;
      private int flagStatus = 0;
      private int agentNum = 0;
      private float agentMass = 10f;
      private Vector3 agentVel;

      // Use this for initialization
      void Start()
      {
            rb = GetComponent<Rigidbody>();
            base.Initialize();
      }

      // Update is called once per frame
      void Update()
      {

      }

      // FixedUpdate is called every physics step
      void FixedUpdate()
      {

            // Si es surt dels límits de l'àrea rep penalització i torna a la
posició inicial
            if (transform.localPosition.y < 0 || transform.localPosition.x > 11 ||
transform.localPosition.x < -11 || transform.localPosition.z > 11 ||
transform.localPosition.z < -11)
            {
                  AddReward(-1f);
                  EndEpisode();
                  area.PlaceAgent(agentNum);
            }

            // Control per si la rotació en els eixos x o z és diferent de 0
```

```
            if (transform.rotation.eulerAngles.x != 0 ||
transform.rotation.eulerAngles.z != 0)
            {
                  // Forcem el valor de la rotació x i z a 0
                  transform.rotation = Quaternion.Euler(0,
transform.rotation.eulerAngles.y, 0);
            }

            // Assignem a un int el color de la bandera
            flagColor = flagRenderer.material.color;
            if (flagColor == Color.white)
            {
                  flagStatus = 0;
            }
            else if (flagColor == Color.blue)
            {
                  flagStatus = 1;
            }
            else if (flagColor == Color.green)
            {
                  flagStatus = 2;
            }
      }

      // Inicialització de l'agent
      public override void Initialize()
      {
            // Assignem el número d'agent a una variable
            if (CompareTag("Agent1"))
            {
                  agentNum = 1;
            }
            else if (CompareTag("Agent2"))
            {
                  agentNum = 2;
            }

            area = GetComponentInParent<Area>();
            flag = area.flag;
            flagRenderer = flag.GetComponent<Renderer>();
      }

      // Cada vegada que comença un episodi
      public override void OnEpisodeBegin()
      {
            // Per entrenar amb CL incrementant la massa
            if (agentNum == 1)
            {
                  agentMass =
Academy.Instance.FloatProperties.GetPropertyWithDefault("agent_mass", 10f);
                  rb.mass = agentMass;
            }
      }
```

```
    // Recollim les variables que tindrà en compte l'agent en el vector
d'observacions
    public override void CollectObservations(VectorSensor sensor)
    {
        // Estat de la bandera (0: sense capturar, 1: en captura per Agent1,
2: en captura per Agent2)
        sensor.AddObservation(flagStatus);

        // Velocitat de l'agent en els eixos x i z
        agentVel = transform.InverseTransformDirection(rb.velocity);

        sensor.AddObservation(agentVel.x);
        sensor.AddObservation(agentVel.z);
    }

    // Interpretem les accions a dur a terme per cada posició del vectorAction
    public override void OnActionReceived(float[] vectorAction)
    {
        var dirToGo = Vector3.zero;
        var rotateDir = Vector3.zero;

        var action = Mathf.FloorToInt(vectorAction[0]);
        if (action == 1)
        {
            //Movem endanvant
            dirToGo = transform.forward * 1f;
        }
        else if (action == 2)
        {
            //Movem enrere
            dirToGo = transform.forward * -1f;
        }
        else if (action == 3)
        {
            //Rotem a la dreta
            rotateDir = transform.up * 1f;
        }
        else if (action == 4)
        {   //Rotem a l'esquerra
            rotateDir = transform.up * -1f;
        }
        //Si l'acció és 0 ens quedem quiets

        //Apliquem la rotació
        transform.Rotate(rotateDir, Time.deltaTime * 150f);

        //Apliquem el moviment
        rb.AddForce(dirToGo * speed, ForceMode.VelocityChange);

        // A cada pas rebem una petita penalització excepte si estem capturant
la bandera
        if (flagStatus != agentNum) AddReward(-1f / maxStep);
    }
```

```
        // Transforma les entrades del teclat en paràmetres del vector accions per
    controlar l'agent manualment
        public override float[] Heuristic()
        {
            float movement = 0;

            if (Input.GetKey(KeyCode.W))
            {
                //Movem endanvant
                movement = 1f;
            }
            else if (Input.GetKey(KeyCode.S))
            {
                //Movem enrere
                movement = 2f;
            }

            if (Input.GetKey(KeyCode.A))
            {
                //Rotem a l'esquerra
                movement = 4f;
            }
            else if (Input.GetKey(KeyCode.D))
            {
                //Rotem a la dreta
                movement = 3f;
            }

            return new float[] { movement };
        }
    }
```

8.2. Area.cs

```
using MLAgents;
using TMPro;
using UnityEngine;

public class Area : MonoBehaviour
{
    // Objectes que hi ha a l'àrea
    public GameObject flag;
    public Agent agent1;
    public Agent agent2;
    private Rigidbody rbAgent1;
    private Rigidbody rbAgent2;

    // Comptadors informatius que es mostren a l'àrea
    public TextMeshPro cumulativeRewardTextAgent1;
    public TextMeshPro cumulativeRewardTextAgent2;

    // Variables usades a les funcions
    private float flagMaxPos = 9.5f;
```

```
    private float x = 0;
    private float z = 0;
    private Vector3 pos;

    // Start is called before the first frame update
    void Start()
    {
        PlaceFlag();   //la bandera neix en una posició aleatòria
        PlaceAgent(1); //col·loquem els agents en posició inicial
        PlaceAgent(2);
    }

    // Update is called once per frame
    void Update()
    {
        // Actualització dels comptadors de recompensa
        cumulativeRewardTextAgent1.text =
agent1.GetCumulativeReward().ToString("0.00");
        cumulativeRewardTextAgent2.text =
agent2.GetCumulativeReward().ToString("0.00");
    }

    // Col·loca la bandera en una posició aleatòria dins el rang [-flagMaxPos,
+flagMaxPos]
    public void PlaceFlag()
    {
        flagMaxPos =
Academy.Instance.FloatProperties.GetPropertyWithDefault("flag_max_position",
9.5f);
        x = Random.Range(-flagMaxPos, flagMaxPos);  //El nostre espai és de 10x10,
limitem a 9,5 per evitar solapaments amb les vores
        z = Random.Range(-flagMaxPos, flagMaxPos);  //El nostre espai és de 10x10,
limitem a 9,5 per evitar solapaments amb les vores
        pos = new Vector3(x, 2, z);
        flag.transform.localPosition = pos;
    }

    // Col·loca els agents en la seva posició inicial
    public void PlaceAgent(int agentNum)
    {
        if (agentNum == 1)
        {
            pos = new Vector3(-5f, 0.5f, 9f);
            rbAgent1 = agent1.GetComponent<Rigidbody>();
            rbAgent1.velocity = Vector3.zero;
            rbAgent1.angularVelocity = Vector3.zero;
            agent1.transform.localPosition = pos;
        }
        else if (agentNum == 2)
        {
            pos = new Vector3(5f, 0.5f, 9f);
            rbAgent2 = agent2.GetComponent<Rigidbody>();
            rbAgent2.velocity = Vector3.zero;
            rbAgent2.angularVelocity = Vector3.zero;
            agent2.transform.localPosition = pos;
```

```
        }
    }
}
```

8.3. Flag.cs

```
using MLAgents;
using System.Collections;
using UnityEngine;
using TMPro;

public class Flag : MonoBehaviour
{
    // Objectes que hi ha a l'àrea
    private Area area;
    private Agent agent1;
    private Agent agent2;
    private Renderer flagRenderer;

    // Comptadors informatius que es mostren a l'àrea
    public TextMeshPro countAgent1text;
    public TextMeshPro countAgent2text;
    public TextMeshPro captureCountdown;

    // Variables de les funcions
    private int countAgent1 = 0;
    private int countAgent2 = 0;
    private IEnumerator coroutine;
    private float flagMaxPos = 9.5f;
    private float x;
    private float z;
    private Vector3 pos;
    private int FlagStatus = 0; //0 no capturada, 1 capturada per Agent1, 2
capturada per Agent2

    // Start is called before the first frame update
    void Start()
    {
        flagRenderer = GetComponent<Renderer>();
        area = GetComponentInParent<Area>();
        agent1 = area.agent1;
        agent2 = area.agent2;
    }

    // Update is called once per frame
    void Update()
    {

    }

    void OnCollisionEnter(Collision collision)
    {
```

```
        // si l'Agent1 colisiona comença a capturar en blau
        if (collision.gameObject.CompareTag("Agent1"))
        {
            if (FlagStatus != 1)
            {
                if (FlagStatus == 2)    // si hi ha captura en curs de l'Agent 2,
l'atura
                {
                    StopCoroutine(coroutine);
                }
                coroutine = Respawn();
                StartCoroutine(coroutine);
                flagRenderer.material.color = Color.blue;
                FlagStatus = 1;
            }
        }
        // si l'Agent2 colisiona comença a capturar en verd
        else if (collision.gameObject.CompareTag("Agent2"))
        {
            if (FlagStatus != 2)
            {
                if (FlagStatus == 1)    // si hi ha captura en curs de l'Agent 1,
l'atura
                {
                    StopCoroutine(coroutine);
                }
                coroutine = Respawn();
                StartCoroutine(coroutine);
                flagRenderer.material.color = Color.green;
                FlagStatus = 2;
            }
        }
    }

    private void OnTriggerStay(Collider other)
    {
        Debug.Log("Collision");
        if (other.gameObject.CompareTag("Untagged"))
        {
            area.PlaceFlag();
        }
    }

    // Funció que desplaça la bandera a una posició aleatòria dins de l'espai de
joc
    void MoveToRandomPosition()
    {
        flagMaxPos =
Academy.Instance.FloatProperties.GetPropertyWithDefault("flag_max_position",
9.5f);
        x = Random.Range(-flagMaxPos, flagMaxPos);  //El nostre espai és de 10x10,
limitem a 9,5 per evitar solapaments amb les vores
        z = Random.Range(-flagMaxPos, flagMaxPos);  //El nostre espai és de 10x10,
limitem a 9,5 per evitar solapaments amb les vores
        pos = new Vector3(x, 2, z);
```

```
            transform.localPosition = pos;
        }

        // Funció que realitza el procés de captura de la bandera
        IEnumerator Respawn()
        {

            // La captura dura 3 segons d'un en un per mostrar el compte enrere a la
UI
            captureCountdown.text = "3...";
            yield return new WaitForSeconds(1);
            captureCountdown.text = "2...";
            yield return new WaitForSeconds(1);
            captureCountdown.text = "1...";
            yield return new WaitForSeconds(1);

            // Un cop passen els 3 segons es desplaça a una nova posició
            area.PlaceFlag();
            captureCountdown.text = "";

            // Suma 1 punt a l'agent que ha capturat
            if (FlagStatus == 1)
            {
                countAgent1++;
                countAgent1text.text = "Agent 1: " + countAgent1.ToString();
                agent1.AddReward(1f);
                agent2.AddReward(-1f);
                agent1.EndEpisode();
                agent2.EndEpisode();
            }
            else if (FlagStatus == 2)
            {
                countAgent2++;
                countAgent2text.text = "Agent 2: " + countAgent2.ToString();
                agent2.AddReward(1f);
                agent1.AddReward(-1f);
                agent1.EndEpisode();
                agent2.EndEpisode();
            }

            // Inicialitza la nova bandera
            flagRenderer.material.color = Color.white;
            FlagStatus = 0;
        }
    }
```

8.4. AgentController_v2.cs

```
using UnityEngine;
using MLAgents;
using MLAgents.Sensors;
using System.Collections;
```

```
public class AgentController_v2 : Agent
{
private Rigidbody rb;
public float speed;
private GameObject flag;
private AreaButton area;
private Renderer flagRenderer;
private Color flagColor;
private int flagStatus = 0;
private int agentNum = 0;
private Vector3 agentVel;
private IEnumerator coroutine;

// Use this for initialization
void Start()
{
      rb = GetComponent<Rigidbody>();
      base.Initialize();
}

// Update is called once per frame
void Update()
{

}

// FixedUpdate is called every physics step
void FixedUpdate()
{

      // Si es surt dels límits de l'àrea rep penalització i torna a la posició
inicial
      if (transform.localPosition.y < 0 || transform.localPosition.x > 11 ||
transform.localPosition.x < -11 || transform.localPosition.z > 11 ||
transform.localPosition.z < -11)
      {
            AddReward(-1f);
            EndEpisode();
            area.PlaceAgent(agentNum);
      }

      // Control per si la rotació en els eixos x o z és diferent de 0
      if (transform.rotation.eulerAngles.x != 0 ||
transform.rotation.eulerAngles.z != 0)
      {
            // Forcem el valor de la rotació x i z a 0
            transform.rotation = Quaternion.Euler(0,
transform.rotation.eulerAngles.y, 0);
      }

      // Assignem a un int el color de la bandera
      flagColor = flagRenderer.material.color;
      if (flagColor == Color.white)
      {
            flagStatus = 0;
```

```
        }
        else if (flagColor == Color.blue)
        {
                flagStatus = 1;
        }
        else if (flagColor == Color.green)
        {
                flagStatus = 2;
        }
}

// Inicialització de l'agent
public override void Initialize()
{
        // Assignem el número d'agent a una variable
        if (CompareTag("Agent1"))
        {
                agentNum = 1;
        }
        else if (CompareTag("Agent2"))
        {
                agentNum = 2;
        }

        area = GetComponentInParent<AreaButton>();
        flag = area.flag;
        flagRenderer = flag.GetComponent<Renderer>();

        // Col·loquem en posició inicial
        area.PlaceAgent(agentNum);
}

// Cada vegada que comença un episodi
public override void OnEpisodeBegin()
{
        // Posem el botó en una posició aleatòria
        area.PlaceButton();
}

// Recollim les variables que tindrà en compte l'agent en el vector d'observacions
public override void CollectObservations(VectorSensor sensor)
{
        // Estat de la bandera (0: sense capturar, 1: en captura per Agent1, 2: en
captura per Agent2)
        sensor.AddObservation(flagStatus);

        // Velocitat de l'agent en els eixos x i z
        agentVel = transform.InverseTransformDirection(rb.velocity);

        sensor.AddObservation(agentVel.x);
        sensor.AddObservation(agentVel.z);

        // Estat del botó (actiu o inactiu)
        sensor.AddObservation(area.buttonActive);
}
```

```
// Interpretem les accions a dur a terme per cada posició del vectorAction
public override void OnActionReceived(float[] vectorAction)
{
	var dirToGo = Vector3.zero;
	var rotateDir = Vector3.zero;

	var action = Mathf.FloorToInt(vectorAction[0]);
	if (action == 1)
	{
		//Movem endanvant
		dirToGo = transform.forward * 1f;
	}
	else if (action == 2)
	{
		//Movem enrere
		dirToGo = transform.forward * -1f;
	}
	else if (action == 3)
	{
		//Rotem a la dreta
		rotateDir = transform.up * 1f;
	}
	else if (action == 4)
	{   //Rotem a l'esquerra
		rotateDir = transform.up * -1f;
	}
	//Si l'acció és 0 ens quedem quiets

	//Apliquem la rotació
	transform.Rotate(rotateDir, Time.deltaTime * 150f);

	//Apliquem el moviment
	rb.AddForce(dirToGo * speed, ForceMode.VelocityChange);

	// A cada pas rebem una petita penalització excepte si estem capturant la
bandera
	if (flagStatus != agentNum) AddReward(-1f / maxStep);
}

// Transforma les entrades del teclat en paràmetres del vector accions per
controlar l'agent manualment
public override float[] Heuristic()
{
	float movement = 0;

	if (Input.GetKey(KeyCode.W))
	{
		//Movem endanvant
		movement = 1f;
	}
	else if (Input.GetKey(KeyCode.S))
	{
		//Movem enrere
		movement = 2f;
```

```
        }

        if (Input.GetKey(KeyCode.A))
        {
                //Rotem a l'esquerra
                movement = 4f;
        }
        else if (Input.GetKey(KeyCode.D))
        {
                //Rotem a la dreta
                movement = 3f;
        }

        return new float[] { movement };
}

void OnTriggerEnter(Collider collision)
{
        // si l'Agent1 colisiona amb el botó s'activa la millora
        if (collision.gameObject.CompareTag("Button"))
        {
                if (area.buttonActive)
                {
                        rb.mass = 100;      // obté més massa
                        speed = 2f;         // duplica la velocitat de moviment
                        coroutine = PowerDown();    // cridem a la rutina que en 10
segons revertirà la millora
                        StartCoroutine(coroutine);
                        area.RestartButton();       // deshabilitem movem el botó de
posició i en 15 segons tornarà a estar actiu
                }
        }
}
IEnumerator PowerDown()
{
        // Al cap de 10 segons restaurem a l'agent els atributs originals
        yield return new WaitForSeconds(10);
        rb.mass = 10;
        speed = 1f;
}
}
```

8.5. AreaButton.cs

```
using MLAgents;
using System.Collections;
using TMPro;
using UnityEngine;

public class AreaButton : MonoBehaviour
{
    // Objectes que hi ha a l'àrea
    public GameObject flag;
```

```
    public GameObject button;
    public Agent agent1;
    public Agent agent2;
    private Rigidbody rbAgent1;
    private Rigidbody rbAgent2;

    // Comptadors informatius que es mostren a l'àrea
    public TextMeshPro cumulativeRewardTextAgent1;
    public TextMeshPro cumulativeRewardTextAgent2;

    // Variables usades a les funcions
    private float flagMaxPos = 9.5f;
    private float x = 0;
    private float z = 0;
    private Vector3 pos;
    private IEnumerator coroutine;
    private IEnumerator reactivation;
    public bool buttonActive = true;

    // Start is called before the first frame update
    void Start()
    {
    }

    // Update is called once per frame
    void Update()
    {
        // Actualització dels comptadors de recompensa
        cumulativeRewardTextAgent1.text =
agent1.GetCumulativeReward().ToString("0.00");
        cumulativeRewardTextAgent2.text =
agent2.GetCumulativeReward().ToString("0.00");
    }

    // Col·loca la bandera en una posició aleatòria dins el rang [-flagMaxPos,
+flagMaxPos]
    public void PlaceFlag()
    {
        flagMaxPos =
Academy.Instance.FloatProperties.GetPropertyWithDefault("flag_max_position",
9.5f);
        x = Random.Range(-flagMaxPos, flagMaxPos);
        z = Random.Range(-flagMaxPos, flagMaxPos);
        pos = new Vector3(x, 2, z);
        flag.transform.localPosition = pos;
    }

    // Col·loca els agents en la seva posició inicial
    public void PlaceAgent(int agentNum)
    {
        if (agentNum == 1)
        {
            pos = new Vector3(-5f, 0.5f, 9f);
            rbAgent1 = agent1.GetComponent<Rigidbody>();
            rbAgent1.velocity = Vector3.zero;
```

```
            rbAgent1.angularVelocity = Vector3.zero;
            agent1.transform.localPosition = pos;
        }
        else if (agentNum == 2)
        {
            pos = new Vector3(5f, 0.5f, 9f);
            rbAgent2 = agent2.GetComponent<Rigidbody>();
            rbAgent2.velocity = Vector3.zero;
            rbAgent2.angularVelocity = Vector3.zero;
            agent2.transform.localPosition = pos;
        }
    }

    // Col·loca el botó en una posició aleatòria
    public void PlaceButton()
    {
        x = Random.Range(-9.5f, 9.5f);
        z = Random.Range(-9.5f, 9.5f);
        pos = new Vector3(x, 0, z);
        button.transform.localPosition = pos;
    }

    // Immobilitza l'agent
    public void FreezeAgent(int agentNum)
    {
        if (agentNum == 1)
        {
            rbAgent1 = agent1.GetComponent<Rigidbody>();
            rbAgent1.velocity = Vector3.zero;
            rbAgent1.angularVelocity = Vector3.zero;
            rbAgent1.constraints = RigidbodyConstraints.FreezeAll;
            coroutine = UnFreeze(1);
            StartCoroutine(coroutine);
        }
        else if (agentNum == 2)
        {
            rbAgent2 = agent2.GetComponent<Rigidbody>();
            rbAgent2.velocity = Vector3.zero;
            rbAgent2.angularVelocity = Vector3.zero;
            rbAgent2.constraints = RigidbodyConstraints.FreezeAll;
            coroutine = UnFreeze(2);
            StartCoroutine(coroutine);
        }
    }

    // Incrementa la massa de l'agent
    public void SuperAgent(int agentNum)
    {
        buttonActive = false;
        reactivation = ButtonReactivation();
        StartCoroutine(reactivation);
        if (agentNum == 1)
        {
            rbAgent1 = agent1.GetComponent<Rigidbody>();
            rbAgent1.mass = 100;
```

```
            coroutine = UnFreeze(1);
            StartCoroutine(coroutine);
        }
        else if (agentNum == 2)
        {
            rbAgent2 = agent2.GetComponent<Rigidbody>();
            rbAgent2.mass = 100;
            coroutine = UnFreeze(2);
            StartCoroutine(coroutine);
        }
    }

    // Retorna a l'agent a la normalitat (mobilitat / massa)
    IEnumerator UnFreeze(int agentNum)
    {
        // Al cap de 10 segons
        yield return new WaitForSeconds(10);
        if (agentNum == 1)
        {
            rbAgent1.mass = 10;
        }
        else if (agentNum == 2)
        {
            rbAgent2.mass = 10;
        }
    }

    // Reactiva el botó al cap de 15 segons
    IEnumerator ButtonReactivation()
    {
        yield return new WaitForSeconds(15);
        buttonActive = true;
    }

    // Desactiva el botó i crida  a la rutina per reactivar-lo
    public void RestartButton()
    {
        buttonActive = false;
        reactivation = ButtonReactivation();
        StartCoroutine(reactivation);
    }
}
```

8.6. FlagButton.cs

```
using MLAgents;
using System.Collections;
using UnityEngine;
using TMPro;

public class FlagButton : MonoBehaviour
{
    // Objectes que hi ha a l'àrea
```

```
    private AreaButton area;
    private Agent agent1;
    private Agent agent2;
    private Renderer flagRenderer;

    // Comptadors informatius que es mostren a l'àrea
    public TextMeshPro countAgent1text;
    public TextMeshPro countAgent2text;
    public TextMeshPro captureCountdown;

    // Variables de les funcions
    private int countAgent1 = 0;
    private int countAgent2 = 0;
    private IEnumerator coroutine;
    private float flagMaxPos = 9.5f;
    private float x;
    private float z;
    private Vector3 pos;
    private int FlagStatus = 0; //0 no capturada, 1 capturada per Agent1, 2
capturada per Agent2

    // Start is called before the first frame update
    void Start()
    {
        flagRenderer = GetComponent<Renderer>();
        MoveToRandomPosition();  //la bandera neix en una posició aleatòria
        area = GetComponentInParent<AreaButton>();
        agent1 = area.agent1;
        agent2 = area.agent2;
    }

    // Update is called once per frame
    void Update()
    {

    }

    void OnCollisionEnter(Collision collision)
    {
        // si l'Agent1 colisiona comença a capturar en blau
        if (collision.gameObject.CompareTag("Agent1"))
        {
            if (FlagStatus != 1)
            {
                if (FlagStatus == 2)    // si hi ha captura en curs de l'Agent 2,
l'atura
                {
                    StopCoroutine(coroutine);
                }
                coroutine = Respawn();
                StartCoroutine(coroutine);
                flagRenderer.material.color = Color.blue;
                FlagStatus = 1;
            }
        }
```

```
        // si l'Agent2 colisiona comença a capturar en verd
        else if (collision.gameObject.CompareTag("Agent2"))
        {
            if (FlagStatus != 2)
            {
                if (FlagStatus == 1)    // si hi ha captura en curs de l'Agent 1,
l'atura
                {
                    StopCoroutine(coroutine);
                }
                coroutine = Respawn();
                StartCoroutine(coroutine);
                flagRenderer.material.color = Color.green;
                FlagStatus = 2;
            }
        }
    }

    private void OnTriggerStay(Collider other)
    {
        Debug.Log("Collision");
        if (other.gameObject.CompareTag("Untagged"))
        {
            MoveToRandomPosition();
        }
    }

    // Funció que desplaça la bandera a una posició aleatòria dins de l'espai de
joc
    void MoveToRandomPosition()
    {
        flagMaxPos =
Academy.Instance.FloatProperties.GetPropertyWithDefault("flag_max_position",
9.5f);
        x = Random.Range(-flagMaxPos, flagMaxPos);  //El nostre espai és de 10x10,
limitem a 9,5 per evitar solapaments amb les vores
        z = Random.Range(-flagMaxPos, flagMaxPos);  //El nostre espai és de 10x10,
limitem a 9,5 per evitar solapaments amb les vores
        pos = new Vector3(x, 2, z);
        transform.localPosition = pos;
    }

    // Funció que realitza el procés de captura de la bandera
    IEnumerator Respawn()
    {

        // La captura dura 3 segons d'un en un per mostrar el compte enrere a la
UI
        captureCountdown.text = "3...";
        yield return new WaitForSeconds(1);
        captureCountdown.text = "2...";
        yield return new WaitForSeconds(1);
        captureCountdown.text = "1...";
        yield return new WaitForSeconds(1);
```

```
        // Un cop passen els 3 segons es desplaça a una nova posició
        MoveToRandomPosition();
        captureCountdown.text = "";
        //area.PlaceRespawn();

        // Suma 1 punt a l'agent que ha capturat
        if (FlagStatus == 1)
        {
            countAgent1++;
            countAgent1text.text = "Agent 1: " + countAgent1.ToString();
            agent1.AddReward(1f);
            agent2.AddReward(-1f);
            agent1.EndEpisode();
            agent2.EndEpisode();
        }
        else if (FlagStatus == 2)
        {
            countAgent2++;
            countAgent2text.text = "Agent 2: " + countAgent2.ToString();
            agent2.AddReward(1f);
            agent1.AddReward(-1f);
            agent1.EndEpisode();
            agent2.EndEpisode();
        }

        // Inicialitza la nova bandera
        flagRenderer.material.color = Color.white;
        FlagStatus = 0;
    }
}
```

8.7. MainMenu.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class MainMenu : MonoBehaviour
{
     public void LoadSceneIAv0() {
        SceneManager.LoadScene("FlagLearning IAv0");
    }
    public void LoadSceneIAv1IAv0()
    {
        SceneManager.LoadScene("FlagLearning IAv1 vs IAv0");
    }
    public void LoadSceneIAv2IAv1()
    {
        SceneManager.LoadScene("FlagLearning IAv2 vs IAv1");
    }
    public void LoadSceneISelfPlay()
    {
```

```
        SceneManager.LoadScene("FlagLearning Selfplay");
    }
    public void LoadSceneVoid()
    {
        SceneManager.LoadScene("FlagLearning Void");
    }
    public void LoadSceneWalls()
    {
        SceneManager.LoadScene("FlagLearning Walls");
    }
    public void LoadScenePw()
    {
        SceneManager.LoadScene("FlagLearningRespawn");
    }
    public void LoadSceneFinal()
    {
        SceneManager.LoadScene("FlagLearningFinal");
    }
    public void Quit()
    {
        Application.Quit();
    }
}
```

8.8. trainer_config.yaml

```
FlagCapture:
    trainer: ppo
    batch_size: 1024
    beta: 5.0e-3
    buffer_size: 10240
    epsilon: 0.2
    hidden_units: 128
    lambd: 0.95
    learning_rate: 3.0e-4
    learning_rate_schedule: linear
    max_steps: 1.0e7
    memory_size: 128
    normalize: false
    num_epoch: 3
    num_layers: 2
    time_horizon: 64
    sequence_length: 64
    summary_freq: 3000
    use_recurrent: false
    vis_encode_type: simple
    reward_signals:
        extrinsic:
            strength: 1.0
            gamma: 0.99
    self_play:
         window: 10
         play_against_current_self_ratio: 0.5
```

```
        save_steps: 3000
        swap_steps: 3000

FlagCaptureWalls:
    trainer: ppo
    batch_size: 1024
    beta: 5.0e-3
    buffer_size: 10240
    epsilon: 0.2
    hidden_units: 128
    lambd: 0.95
    learning_rate: 3.0e-4
    learning_rate_schedule: linear
    max_steps: 1.0e7
    memory_size: 128
    normalize: false
    num_epoch: 3
    num_layers: 2
    time_horizon: 64
    sequence_length: 64
    summary_freq: 3000
    use_recurrent: false
    vis_encode_type: simple
    reward_signals:
        extrinsic:
            strength: 1.0
            gamma: 0.99
    self_play:
         window: 10
         play_against_current_self_ratio: 0.5
         save_steps: 3000
         swap_steps: 3000
    curiosity:
        strength: 0.02
        gamma: 0.99
        encoding_size: 256
```

8.9. flag_capture.yaml

```
FlagCapture:
  measure: reward
  thresholds: [0.2, 0.4, 0.6, 0.8]
  min_lesson_length: 100
  signal_smoothing: true
  parameters:
    flag_max_position: [0.0, 2.5, 5.0, 7.5, 9.5]
    agent_mass: [20, 17.5, 15, 12.5, 10]
```