Progetto ICon

Leonardo Menna 700755

Tommaso Narracci 699183

Link repository Github

INDICE

1 - INTRODUZIONE

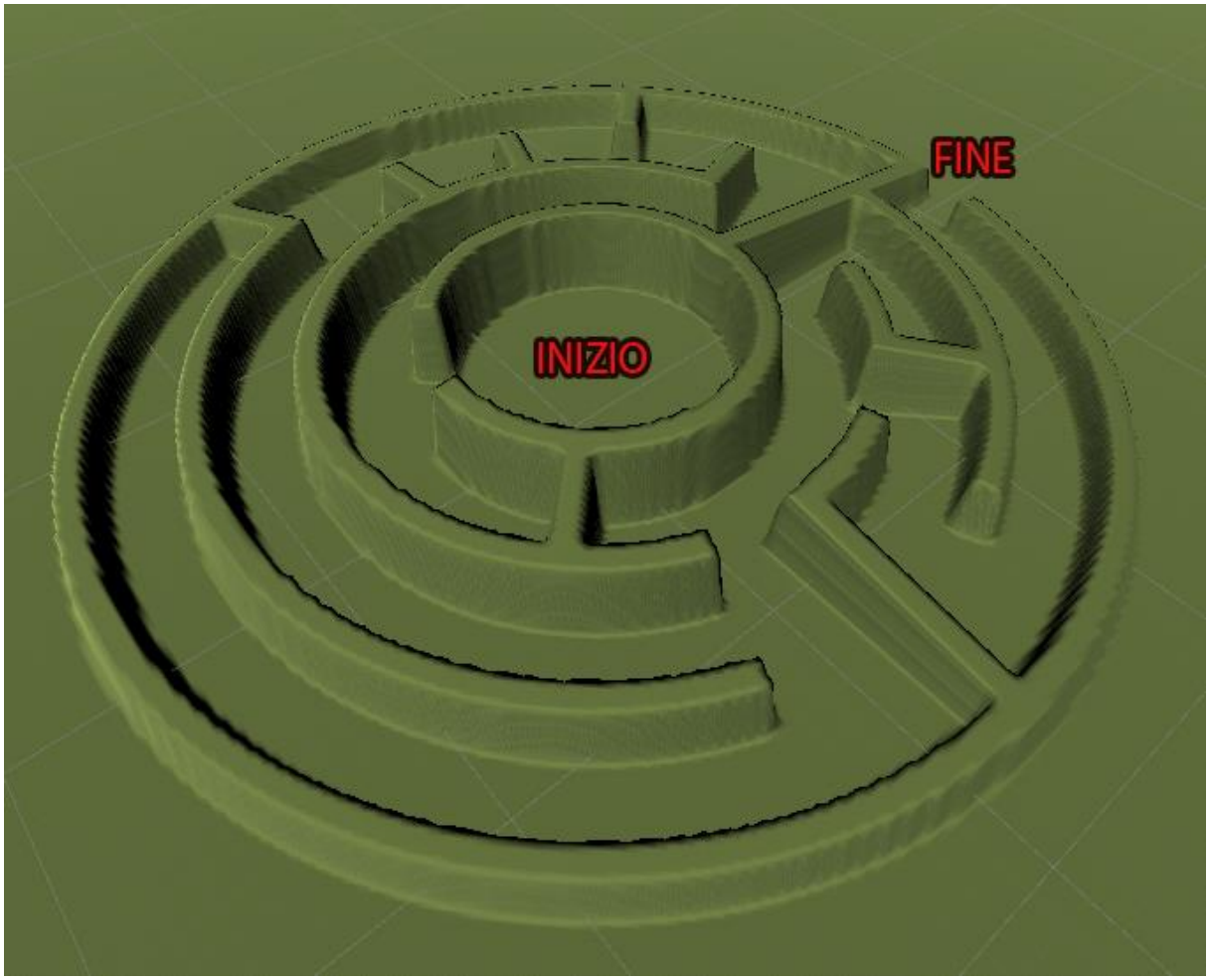
In questo progetto abbiamo realizzato un intelligenza artificiale che opera in un contesto videoludico.

Per la creazione del progetto ci siamo serviti di Unity, questo software è utilizzato per lo sviluppo di videogiochi e si basa sul linguaggio di programmazione del C Sharp. Per rendere possibile ciò che avevamo in mente di fare abbiamo utilizzato una libreria molto spesso usata per il Machine Learning su Unity : ML-Agents.

ML-Agents permette di rendere più semplice, attraverso algoritmi e componenti di Unity, la creazione di agenti che sfruttino il Reinforcement Learning (in specifico si serve dell'algoritmo PPO) per acquisire esperienza e diventare sempre più bravi e efficienti ad ogni tentativo. Questo grazie ad un sistema di reward/punishment, che, dopo un po' di tentativi, farà capire all'agente le azioni giuste da intraprendere ad ogni step per massimizzare i reward, tutto ciò analizzando le osservazioni fornite ad esso sull'ambiente circostante. Inizierà quindi nei primi tentativi ad eseguire azioni casuali esplorando quindi varie soluzioni e, ad ogni tentativo, scandito dall'inizio di un nuovo "episodio", inizierà a fare azioni che massimizzino il reward per ogni step, senza mai dimenticarsi di continuare ad esplorare nuove strategie (questo grazie al parametro "curiosity").

2 - IDEA DEL PROGETTO

L’idea alla base del progetto è stata realizzare un intelligenza artificiale che potesse trovare la via di uscita all’interno di un labirinto e allo stesso tempo fosse capace di difendersi dagli oppositori durante il percorso.



Inizialmente ci siamo dedicati alla creazione del labirinto, la parte centrale e il punto in cui il nostro agente inizia il percorso.

Gli oppositori sono stati programmati per inseguire l’agente all’interno del labirinto, con lo scopo di prenderlo, e se riescono, l’agente perde ed è costretto a rincominciare da capo.

Per far sì che l’agente apprendesse in base al contesto è stato allenato per molte ore.

Gli obiettivi che il nostro agente deve ottenere quindi sono:

-Eliminare gli oppositori

-Concludere il labirinto

Per conseguire gli obiettivi abbiamo definito un sistema di premi e punizioni in base a quanto detto prima riguardo il Reinforcement Learning.

L’agente ottiene dei premi se:

-colpisce un oppositore e lo fa nel minor tempo possibile

-si dirige verso il percorso corretto

L’agente ottiene delle punizioni se:

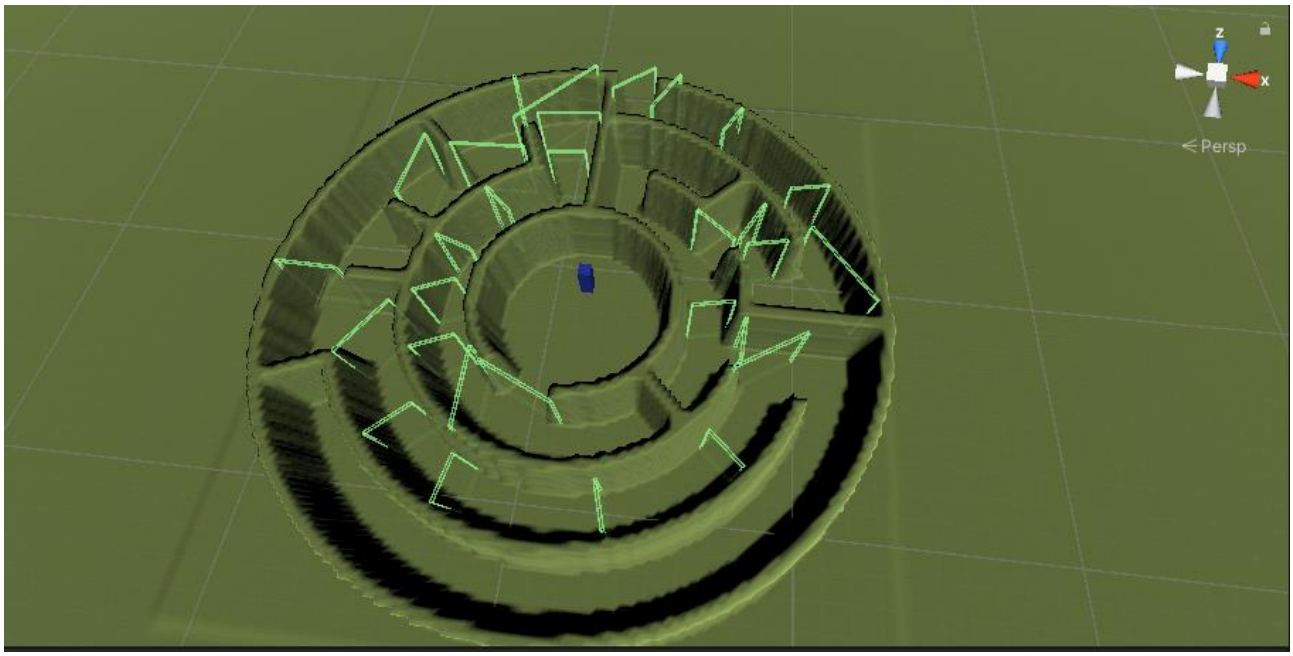
-manca il colpo

-Tocca i muri interni del labirinto

-Passa un tempo stabilito e non ha ancora trovato il percorso corretto

3 - LABIRINTO

Per quanto riguarda la parte del labirinto, per fare in modo che l’agente riesca a capire quale è il percorso corretto, abbiamo inserito dei muri invisibili e oltrepassabili all’interno del labirinto.



Abbiamo utilizzato il metodo OnCollisionEnter() che è un metodo che viene richiamato quando un corpo rigido oltrepassa un altro corpo rigido.

Questo metodo è presente all'interno dello script muro_trigger e verifica tramite un controllo che quando il muro viene oltrepassato dal nostro agente, quest'ultimo ottiene un premio.

Dato che il metodo della collisione viene richiamato ogni volta che l'agente oltrepassa il muro, riusciva ad ottenere i premi semplicemente spostandosi avanti e indietro facendo una collisione.

Per evitare questa situazione il muro viene disattivato, ovvero non può fornire più premi in alcun modo.

Poi ad ogni Episodio, tutti i muri ritornano ad essere attivi.

In questo modo l'agente riesce durante l'addestramento a capire quale è il percorso corretto.

Come abbiamo accennato prima l'agente ottiene anche delle punizioni, infatti ogni volta che l'agente non riesce a trovare il percorso corretto, dopo un tempo definito riceve una penalità e in questo modo viene motivato a trovare il percorso corretto nel minor tempo possibile, quindi ogni volta che l'agente non trova un muro, che indica il percorso corretto nell'arco di 100 secondi, viene penalizzato e farà ripartire l'episodio.

Oltre a questa penalità, può capitare che l'agente si blocchi nei muri interni del labirinto, per ovviare a questo problema abbiamo riutilizzato il metodo OnCollisionEnter(), questa volta all'interno dello script associato all'agente.

Tramite questo metodo l'agente viene penalizzato quando tocca un muro del labirinto e rimane bloccato. Se l'agente rimane bloccato per oltre 10 secondi nel muro riceverà una penalità e ripartirà l'episodio.

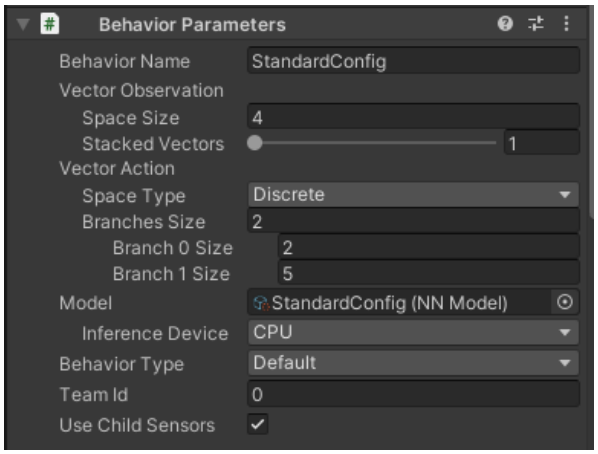
Sempre in OnCollisionEnter() abbiamo fatto un controllo, se un oppositore fa collisione con l'agente, questo porta a ricevere una penalità e allo stesso tempo comporta la fine dell'episodio, quindi l'agente è costretto a ripartire da capo.

4 - SPARO

La parte dello "sparo" è stata costruita passo per passo, questo per capire bene l'utilizzo della libreria e per capire se si stesse andando nella direzione giusta. Siamo partiti semplicemente con l'agente e un nemico uno di fronte all'altro. Durante il training, l'agente doveva quindi solamente capire che doveva sparare più velocemente possibile, appena aveva disponibile il colpo in

canna. Quindi abbiamo collezionato osservazioni sulla variabile bool shotReady e abbiamo assegnato un reward negativo (-1/MaxStep) per far comprendere all'agente di terminare l' episodio più in fretta possibile. Le osservazioni sono conservate in un vettore che ha tante celle quanto il numero di osservazioni.Nel nostro caso il vettore ha 4 celle:1 per l'osservazione di shotReady e 3 (x,y,z) per il vettore che indica la direzione che il nostro agente sta seguendo. Ogni volta che l'agente uccide il nemico riceve una ricompensa e riparte l'episodio. L'agente è capace di prendere decisioni con i metodi Heuristic, dove vanno inseriti i "comandi" impartibili all'agente(es. W per spostarsi in avanti), e OnActionReceived, dove andranno implementate le azioni causate da un comando dato in input(se ha premuto W, vai avanti).

I 2 metodi ricevono come parametro un vettore che può essere discreto o continuo. Noi abbiamo usato un vettore discreto con 2 celle: la 1° che può contenere 2 valori, ovvero 0 e 1 a seconda se si vuole sparare o no, e la 2° con 5 valori, a seconda del movimento o della rotazione.

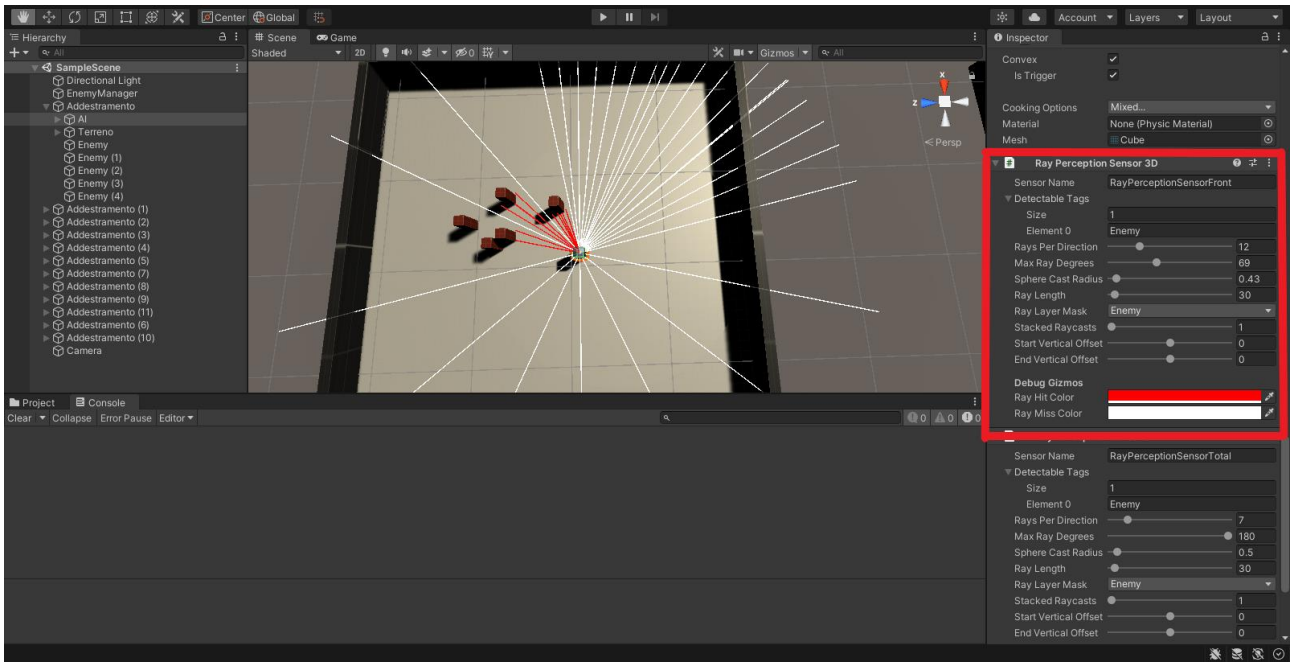


Questi metodi durante la fase di training verranno chiamati in sequenza ogni tot. step(indicabili manualmente) dallo script

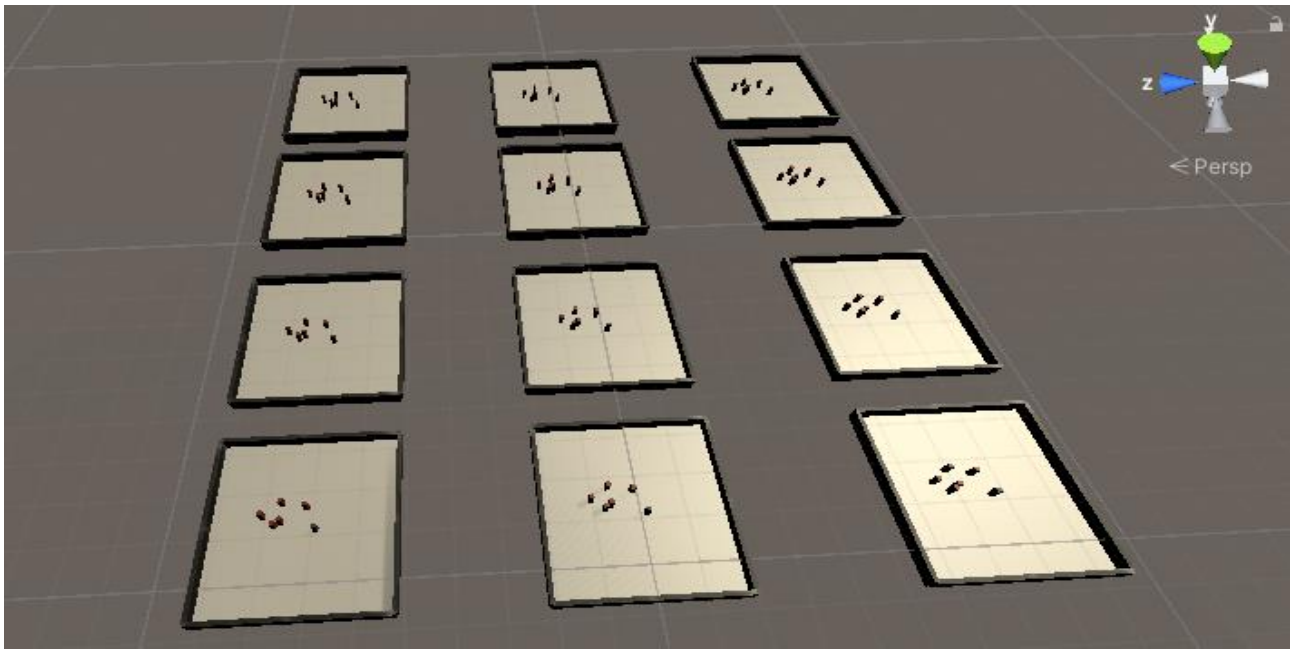
Decision Requester che è fornito direttamente da ML Agents, che sarà responsabile di prendere decisioni.

Successivamente sono stati implementati i movimenti dell'agente lo spawn randomico di un nemico sull'asse Z(in orizzontale). Siamo poi passati ad aggiungere la rotazione all'agente e lo spawn randomico del nemico questa volta sull' asse X e Z.

Infine abbiamo inserito più nemici nell'arena che inseguivano il nostro agente(tramite il componente NavMeshAgent di Unity).Abbiamo anche aggiunto una penalità per i colpi mancati e, un componente fondamentale, i RayPerceptionSensor 3D forniti da MLAgents:questi sensori sparano tot. raggi in tot. direzioni(indicate in gradi) e, se impostati correttamente(attraverso il campo Detectable Tags) collezioneranno automaticamente osservazioni sull'ambiente intorno all'Agente in base ai Tag degli oggetti che indichiamo di osservare(gli Enemy hanno tag Enemy, se in Detectable Tags inseriamo Enemy il sensore potrà rilevarli e collezionare informazioni sulla loro posizione).



Una tecnica molto importante che abbiamo utilizzato per velocizzare i tempi di training è stata quella di replicare gli ambienti per più volte,infatti ML-Agents permette ad Agenti che condividono lo stesso Behaviour di collezionare osservazioni e reward tutti insieme, velocizzando quindi di molto il processo di allenamento(vedi qui per maggiori info).



5 - CONGIUNZIONE DELLE 2 PARTI E ADDESTRAMENTO

Dopo aver appurato il funzionamento di entrambe le parti abbiamo proceduto ad unirle e adattarle per funzionare bene insieme. Successivamente abbiamo fatto partire l'allenamento completo.

Per l'addestramento ci siamo serviti del CMD di Anaconda, dove scaricando i pacchetti relativi ad ML-Agents(compresi altri tool come Pytorch ecc…breve tutorial) e andando nella directory del progetto, usando il comando:

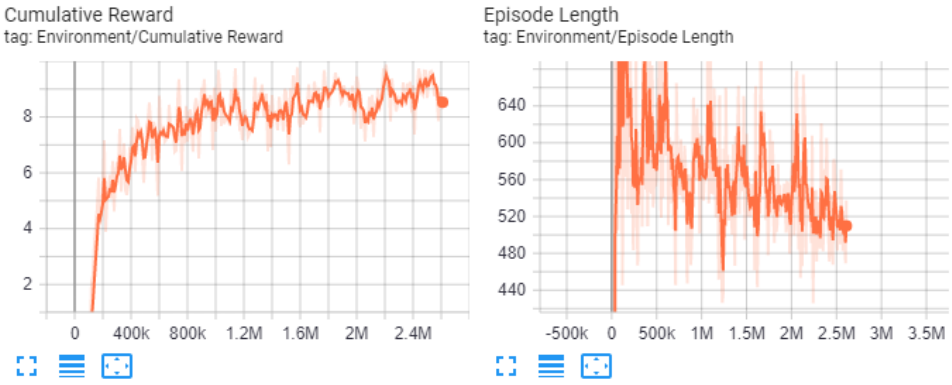
"mlagents-learn -logdir "DIRECTORY CHE CONTIENE FILE.YAML" –run-id="NOME RUN"

e premendo Play nell'Editor di Unity,partiva il vero e proprio Training.Nel file .yaml sono presenti i vari parametri del training.(Significati parametri)

L'addestramento finale è durato oltre 3 ore.

Ecco i risultati dell'addestramento forniti da tensorboard dopo aver usato il comando:

tensorboard dev upload –logdir "DIRECTORY RISULTATO"



6 – CONCLUSIONI

E’ stato interessantissimo osservare come, al passare dei tentativi, il nostro agente diventava sempre migliore nei compiti che doveva effettuare. Nonostante le difficoltà nel realizzare questa intelligenza artificiale il risultato, a parer nostro, è stato sorprendente perché fa capire quanto possono essere efficaci queste tecniche di Machine Learning e quanti compiti complessi possono imparare a svolgere in maniera perfetta grazie all’esperienza ottenuta nei vari tentativi, proprio come facciamo noi esseri umani.

Link Github del progetto Completo.