Alma Mater Studiorum · Università di Bologna

Campus di Cesena
Scuola di Ingegneria
CORSO DI LAUREA MAGISTRALE IN INGEGNERIA E SCIENZE INFORMATICHE

REINFORCEMENT LEARNING: UN CASO DI STUDIO NELL'AMBITO DELLA ANIMAL-AI OLYMPICS

Tesi in: Machine Learning

Relatore:
Prof. DAVIDE MALTONI
Correlatore:
DOTT. VINCENZO LOMONACO

Presentata da:
DIEGO PERGOLINI

ANNO ACCADEMICO 2018-2019
III SESSIONE

Materia ed energia erano terminate e, con esse, lo spazio e il tempo. Perfino AC esisteva unicamente in nome di quell'ultima domanda alla quale non c'era mai stata risposta dal tempo in cui un assistente semi-ubriaco, dieci trilioni d'anni prima, l'aveva rivolta a un calcolatore che stava ad AC assai meno di quanto l'uomo stesse all'Uomo.
Tutte le altre domande avevano avuto risposta e, finché quell'ultima non fosse stata anch'essa soddisfatta, AC non si sarebbe forse liberato della consapevolezza di sé.
Tutti i dati raccolti erano arrivati alla fine, ormai. Da raccogliere, non rimaneva più niente.
Ma i dati raccolti dovevano ancora essere correlati e accostati secondo tutte le relazioni possibili.
Un intervallo senza tempo venne speso a far questo.
E accadde, così, che AC scoprisse come si poteva invertire l'andamento dell'entropia.
Ma ormai non c'era nessuno cui AC potesse fornire la risposta all'ultima domanda. Pazienza! La risposta - per dimostrazione - avrebbe provveduto anche a questo.
Per un altro intervallo senza tempo, AC pensò al modo migliore per riuscirci. Con cura, AC organizzò il programma.
La coscienza di AC abbracciò tutto quello che un tempo era stato un Universo e meditò sopra quello che adesso era Caos.
Un passo alla volta, così bisognava procedere.
LA LUCE SIA! disse AC.
E la luce fu …

Isaac Asimov, *L'ultima domanda*, 1956

ABSTRACT

Keywords: Reinforcement Learning, Machine Learning, Intelligenza artificiale, PPO, Reward Shaping, Animal-AI Olympics, Depth-Estimation, Model-Free RL.

Il campo del Reinforcement Learning (RL) ha ottenuto risultati notevoli negli ultimi anni, proponendo modelli che, fra i vari risultati, sono riusciti a raggiungere e superare il livello umano in giochi delle vecchie console Atari, nel complesso ed antichissimo gioco del GO ed in videogiochi strategici in tempo reale (RTS) quali Starcraft[47] e Dota2[29]. La problematica principale sta però nel fatto che questi approcci sono spesso incapaci di generalizzare ed adattarsi a sfide molto diverse da quelle su cui sono stati addestrati. La domanda a cui cerca di rispondere questo lavoro è quindi la seguente: possono le tecniche del RL affrontare sfide complesse in cui sono richieste diverse abilità cognitive per superare brillantemente situazioni anche molto diverse tra loro come potrebbe fare un animale o ancor di più un umano?
Ovviamente per dimostrare questo si dovrà disporre di un benchmark robusto e significativo e proprio a questo scopo nasce la competizione Animal-AI Olympics[3]. Gli organizzatori di questa competizione hanno messo a disposizione un ambiente in cui l'agente intelligente si muoverà ed interagirà con varie tipologie di oggetti, con l'obiettivo primario di procurarsi del cibo. La competizione include 10 categorie di test, ognuna atta a dimostrare l'effettivo possesso di una determinata capacità cognitiva, quale ad esempio, la capacità di avere preferenze, di capire la fisica di un ambiente o di sapere utilizzare il ragionamento causale. L'agente che dovrà affrontare queste sfide avrà due soli input sensoriali, ovvero visione monoculare e percezione della propria velocità.
In questo lavoro si è utilizzato PPO[38] come algoritmo di RL di riferimento, semplificando però i dati della visione monoculare considerando solo i colori legati a stimoli positivi e negativi e dividendo l'immagine in quadranti, in modo da fornire dati di input più significativi rispetto alle immagini RGB di partenza, contenenti molti elementi di confusione per l'agente. Al fine di ottenere i migliori risultati possibili, il processo di addestramento è stato coadiuvato da tecniche quali curriculum learning e reward shaping, unitamente alla definizione di scenari il più possibile istruttivi per l'agente. Tramite la tecnica del curriculum learning si è voluto far imparare all'agente innanzitutto la relazione fra oggetti e ricompensa/penalità ricevuta, per poi addestrarlo ad ottenere le stesse ricompense in ambiente più complessi.
Il reward shaping, in questo caso basato sulle velocità percepite dall'agente, è stato utile per spingere l'algoritmo a trovare strategie intelligenti

per esplorare al meglio l'ambiente al fine di ottenere le ricompense presenti in esso.
Si sono poi condotte delle sperimentazioni su approcci multi-modello, analizzando i risultati ottenuti e mostrando al contempo i limiti ed i possibili sviluppi futuri.
Sono state inoltre condotte indagini preliminari riguardo alla realizzazione di approcci per il mapping di un ambiente a partire da sole immagini monoculari.
Il risultato finale ottenuto nella competizione è stato un 9°posto su un totale di 64 partecipanti, con un punteggio di 32.6% di test superati, ovvero una decina di punti in meno rispetto alla prima posizione in classifica, ma si è ottenuto un premio per il miglior punteggio sulla categoria 2 (relativa alla capacità di fare la scelta più vantaggiosa nello scegliere il cibo). I risultati dimostrano la bontà delle scelte intraprese ma evidenziano anche come i punteggi più alti siano stati ottenuti nelle categorie che richiedono di esibire abilità cognitive più basilari, motivo per cui il percorso per arrivare ad un modello davvero generale in tutte le casistiche è ancora lungo.

INTRODUZIONE

Fin dagli albori della storia, l'essere umano si è sempre chiesto come funzionasse la propria mente e quali fossero i principi di funzionamento che gli hanno permesso di diventare la specie dominante sul pianeta Terra. Miti, religioni, correnti filosofiche e scienziati hanno a lungo cercato, e stanno tuttora cercando, una risposta a questi quesiti fondamentali. Tanti interrogativi riguardanti le scienze cognitive sono stati risolti, ma tanti ancora richiedono una risposta certa, data la complessità della materia. L'umanità però negli anni '50 dello scorso secolo si è posta un nuovo obiettivo: emulare la propria intelligenza in un corpo sintetico.
Nel 1965 Herbert Simon annunciava: "*le macchine saranno in grado, entro vent'anni, di svolgere qualsiasi lavoro un uomo possa fare*"[41].
Ad oggi, quasi nel 2020, la ricerca mondiale è concorde nel fatto che l'ottimismo di quegli anni fu esagerato e il campo dell'intelligenza artificiale (AI) è perciò rimasto, tra alti e bassi (i cosiddetti inverni dell'AI), confinato nei laboratori di ricerca, senza godere di troppo risalto, ma tutto questo fino ad una decina di anni fa.
Nell'ultimo decennio, con un ritmo sempre più impressionante, la ricerca ha sfornato nuovi metodi ed ottenuto risultati impressionanti. L'uomo ad oggi, non è più il miglior giocatore di scacchi o di Go[39], l'AI vince tornei di e-sport quali StarCraft[47] o Dota2[29]. Si è quindi raggiunto l'obiettivo di eguagliare e superare l'intelligenza umana? Purtroppo o per fortuna tutti i risultati citati sono frutto di modelli specifici per il task di applicazione, essi peccano quindi della capacità di generalizzare, cosa in cui gli esseri umani eccellono.
In realtà la capacità di generalizzare della maggior parte dei modelli di AI non è in grado di competere nemmeno con quella dimostrata dagli animali, perché allora non provare a porsi la sfida di eguagliare le capacità cognitive degli animali? In fondo se si supererà questo step l'obiettivo finale sarà molto più a portata di mano.
Proprio a questo scopo nasce quindi la competizione Animal-AI Olympics[3], la quale si propone di utilizzare i test cognitivi, che negli ultimi 100 anni sono stati sviluppati per verificare le abilità cognitive degli animali, per addestrare e validare modelli di AI.
In questo caso non si vuole testare un agente in uno specifico caso, bensì gli organizzatori metteranno a disposizione un'arena in cui l'agente si muoverà ed una certa lista di abilità cognitive da dimostrare. I test che verranno predisposti saranno nascosti e concepiti in modo che ognuno di essi possa dimostrare il possesso di una particolare capacità cognitiva.
Questa competizione sarà l'occasione per testare un campo dell'intelligenza artificiale che negli ultimi anni ha ricoperto un ruolo centrale nella ricerca: il **Reinforcement Learning** (RL). Sarà particolarmente

interessante vedere se il RL sarà in grado di dimostrare tutte le capacità cognitive possedute dagli animali più intelligenti. La tesi in questione si propone quindi l'obbiettivo di studiare l'efficacia di questo approccio in un ambito applicativo davvero sfidante quale quello della competizione sopra citata, utilizzando un algoritmo allo state dell'arte come PPO.
Si partirà introducendo nel capitolo 1 i concetti chiave del Reinforcement Learning, andando poi a fare una distinzione fra i vari approcci esistenti in questo ambito. Nel capitolo 2 si approfondirà un algoritmo allo stato dell'arte, PPO[38], mostrandone i principi e le applicazioni. Verrà poi discussa in dettaglio la competizione Animal-AI Olympics nel capitolo 3, spiegando tutti gli aspetti che la riguardano e che possono essere utili per affrontarla. Nei capitoli 4 e 5 verranno mostrati gli approcci con cui si è affrontata la competizione, per poi presentare i risultati ottenuti nel capitolo 6 e trarre le dovute conclusioni e possibili sviluppi futuri nel 7.

INDICE

ELENCO DELLE FIGURE

ELENCO DELLE TABELLE

ACRONIMI

RL Reinforcement Learning

PPO Proximal Policy Optimization

NN Neural Network

RNN Recurrent Neural Network

CNN Convolutional Neural Network

LSTM Long Short Term Memory

AAIO Animal-AI Olympics

RTS Real Time Strategy

ML Machine Learning

MSE Mean Squared Error

MDP Markov Decision Process

1

REINFORCEMENT LEARNING

Obiettivo di questo capitolo è di fornire un introduzione al tema del **reinforcement learning**, paradigma di apprendimento che verrà utilizzato come elemento fondamentale del progetto. Al fine di comprendere chiaramente le tematiche affrontate, verranno presentati prima gli elementi chiave comuni a tutti gli approcci di questa famiglia di algoritmi, per poi definire una tassonomia dei metodi esistenti allo stato dell'arte. Particolare attenzione verrà dedicata alla categoria degli algoritmi di Model-Free RL, in quanto costituiscono l'ambito di studio principale nella materia.

1.1 INTRODUZIONE

L'essenza dell'apprendimento è la capacità di calibrare degli elementi di un sistema complesso in risposta ad un feedback, tipicamente esterno ma anche auto-generato. Che tale sistema sia umano, animale, meccanico o puramente software, l'apprendimento è fare tesoro dell'esperienza acquisita, modificando un comportamento a seguito di un'interazione. Sebbene si possa ragionare a lungo sui vari significati del termine, spaziando su varie scienze, in questo capitolo ci si riferirà al termine apprendimento nell'accezione informatica.
Nel campo dell'apprendimento automatico si cerca di rispondere alla seguente domanda:

> Come possiamo costruire sistemi informatici che migliorano automaticamente con l'esperienza e quali sono le leggi fondamentali che governano tutti i processi di apprendimento?[22]

Tra i vari approcci per rispondere a questa domanda si fa spesso una classificazione di base in:

- **Apprendimento supervisionato**: il training del software viene effettuato presentando per ogni input mostrato l'etichetta relativa all'output che dovrebbe corrispondere ad esso. Il sistema deve essere perciò in grado di individuare relazione fra input ed output, relazioni che però non è scontato che esistano, in taluni casi potrebbero essere soltanto co-occorrenze.
- **Apprendimento non supervisionato**: è un tipo di apprendimento più complesso in cui il sistema analizza i dati alla ricerca

di relazioni nascoste. In questo caso i dati non sono etichettati, e devono essere auto-organizzati in base alle loro caratteristiche.

- **Apprendimento per rinforzo**: scopo di questa metodologia è di capire quali azioni sono da eseguire in conseguenza di uno stimolo e delle altre già eseguite. Si tratta quindi di imparare dall'interazione con l'ambiente (fisico o virtuale), al fine di massimizzare la ricompensa ottenuta. Si può vedere questa quantità come la misura di quanto bene stia facendo l'agente software rispetto a quanto da noi richiesto.
  Questo è il caso in cui non è possibile definire formalizzare il tutto in termini di precisi dati di ingresso/uscita, per spiegare l'idea alla base molto spesso si fa riferimento alla sua applicazione psicologica, ovvero che il dare una ricompensa immediatamente dopo una buona azione aumenta la probabilità che essa si verifichi di nuovo.

1.2 CONCETTI CHIAVE

Prima di iniziare ad enucleare i vari aspetti fondanti del reinforcement learning è utile partire dall'idea di base che sta dietro a questo approccio, come egregiamente descritta da Sutton e Barto:

> *L'idea per cui apprendiamo interagendo con il nostro ambiente è probabilmente la prima che viene in mente quando si pensa alla natura dell'apprendimento. Quando un bambino gioca, agita le braccia, o guarda in giro, non ha un qualcuno che gli dica cosa fare, ma ha una connessione sensomotoria diretta con l'ambiente che lo circonda. L'esercizio di questa connessione produce una grande quantità di dati: relazioni causa ed effetto, conseguenze delle proprie azioni e sequenze di azioni per raggiungere certi obbiettivi. Nel corso delle nostre vite, tali interazioni sono senza dubbio una fonte importante di conoscenza del nostro ambiente e di noi stessi. Se stiamo imparando a guidare un'auto o per tenere una conversazione, siamo profondamente consapevoli di come il nostro ambiente risponde a ciò che facciamo e perciò cerchiamo di influenzare ciò che accade attraverso il nostro comportamento.*[42]

A partire da questa idea, un algoritmo di RL si occupa di imparare cosa fare, ovvero mappare delle situazioni a delle azioni, in modo da massimizzare un segnale di reward numerico. La potenza di questo approccio è che non dobbiamo direttamente descrivere quali azioni l'agente deve eseguire, bensì è esso stesso che deve scoprire quali di esse porterà a massimizzare il reward ricevuto.

1.2.1 *Ciclo d'interazione agente-ambiente*

Come anticipato, il RL è fondato sul concetto d'interazione tra agente ed ambiente. L'ambiente è lo spazio, fisico o virtuale, in cui l'agente vive e con cui interagisce. Ad ogni passo di questo loop potenzialmente infinito l'**agente**, il quale possiede una rappresentazione parziale o completa dello stato del mondo, decidere quale **azione** compiere. L'**ambiente** a sua volta produrrà un nuovo **stato** e, cosa fondamentale, una **ricompensa** (o una penalità). I cinque elementi appena menzionati costituiscono la

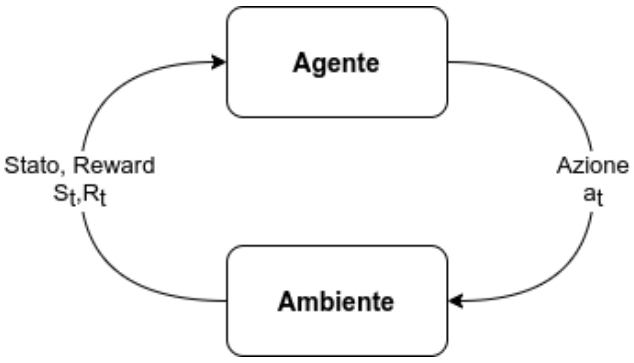


Figura 1: Ciclo di interazione agente-ambiente

base fondante del paradigma del RL. L'obiettivo dell'agente è infatti quello di massimizzare la somma delle ricompense nel tempo, detto anche **return**. Questo segnale numerico può essere visto come una misura quantitativa di quanto bene, o male, sta facendo l'agente. Nelle prossime sezioni verrà spiegata meglio l'importanza della funzione di reward, ma è fondamentale capire fin da subito che strutturare questo segnale non è banale e richiede grande esperienza e ragionamento. Infatti, queste tecniche cercano in ogni modo di massimizzare il return, ma se non si è ragionato correttamente a come strutturare la funzione che lo fornisce, si potrebbero ottenere comportamenti diversi da quelli desiderati.
In definitiva l'agente cercherà di capire come le proprie azioni, compiute in un determinato stato osservato, influenzano l'ottenimento del reward. Ovviamente quello appena presentato è un modello, nei casi in cui il reinforcement learning viene applicato a casi del mondo reale e fisico per ottenere questo loop bisognerà discretizzare e decidere ogni quanto campionare l'ambiente, quindi sebbene esista un modello di riferimento, bisognerà poi fare una seria analisi per capire come calarlo nella realtà.

1.2.2 *Stati ed osservazione dell'ambiente*

Nella sezione precedente è stato introdotto il concetto di stato, in realtà però bisogna fare una chiara distinzione fra il concetto di stato ed il concetto di observation:

- Lo **stato** s è la completa descrizione del mondo, comprendente di tutti gli aspetti.
- L'**observation** o è una descrizione parziale dello stato, che quindi potrebbe omettere delle informazioni. Risulta ovvio come più ci si confronta con ambienti simili alla realtà e più diventa praticamente impossibile descrivere interamente lo stato del mondo. In ogni caso d'ora in avanti se si farà riferimento allo stato, come spesso accade nella letteratura, si deve pensare all'observation o, che è tutto ciò che l'agente può percepire. Definire correttamente come strutturare o è di fondamentale importanza per ottenere risultati soddisfacenti.

In relazione a questi due concetti possiamo perciò definire un ambiente **completamente osservabile** se o coincide con s, mentre se non tutti gli elementi di s sono rappresentabili in o allora si dice **parzialmente osservabile.**
Nel primo caso possiamo formalizzare il tutto come un processo decisionale di Markov (MDP), nel secondo invece come un MDP parzialmente osservabile, ovvero un POMDP.

1.2.2.1 *Markov Decision Process*

Un MDP è definito come una quadrupla $< S, A, P_a, R_a >$ così composta:

- S l'insieme degli stati
- A l'insieme delle azioni
- $P_a(s, s')= P_r(s_{t+1} = s'|s_t = s, a_t = a)$ è la probabilità che l'azione a compiuta nello stato s al tempo t porterà allo stato s' al tempo $t + 1$
- $R_a(s, s')$ è la ricompensa ottenuta passando dallo stato s allo stato s' attraverso l'azione a.

Uno stato S_t è detto Markoviano, cioè gode della proprietà di Markov[19] se:

$$P[S_{t+1}|S_t] = P[S_{t+1}|S_1, ..., S_t] \tag{1}$$

O in altre parole se **il futuro è indipendente dal passato dato il presente**, ciò implica che tutte le informazioni rilevanti del passato sono catturate nello stato attuale.

1.2.3 *Spazio delle azioni e policy dell'agente*

1.2.3.1 *Spazio delle azioni*

Quando si parla di azioni che l'agente può compiere è bene parlare di **spazio delle azioni**, ovvero l'insieme delle possibile mosse effettuabili in un determinato ambiente.

In alcuni casi questo spazio è **discreto**, ovvero esiste un numero finito di azioni che l'agente può eseguire.
In altri casi invece, tra i quali possiamo annoverare il controllo di un robot nel mondo fisico, lo spazio delle azioni è **continuo**.
E' bene tenere in mente fin da subito questa distinzione, perché essa influenza il funzionamento degli algoritmi di RL.

1.2.3.2 *Policy di un agente*

Il concetto di policy è il cardine del RL, infatti essa descrive la strategia di comportamento dell'agente, definendo quindi il mapping tra stato percepito ed azione da eseguire. La policy può essere descritta come una funzione, una tabella o tramite metodi più complessi, come verrà mostrato nei capitoli successivi.
Essenzialmente una policy π indica quale azione eseguire nello stato s_t.
Si può poi distinguere fra due tipi di policy:

- **Deterministiche**: ovvero $a = \pi(s_t)$, non c'è quindi incertezza sul fatto che ogni volta che l'agente si troverà allo stato s eseguirà sempre la stessa azione.
- **Stocastiche:** ovvero $P[A_t = a|S_t = s] = \pi(a|s)$, in questo caso invece l'azione che verrà eseguita dipende da una distribuzione di probabilità.

1.2.4 *Funzione di reward e return atteso*

Come già anticipato il segnale di reward R_t rappresenta numericamente quanto bene sta facendo l'agente al tempo t. Facendo un parallelo biologico lo si può immaginare come "gratificazione" se questo numero è positivo o "punizione" se è negativo. Risulta perciò evidente come obiettivo di un addestramento è quello di imparare quali stati e/o quali azioni compiute portino a massimizzare le ricompense ottenute, aggiustando quindi la policy di conseguenza.
Nell'apprendimento per rinforzo si fa l'ipotesi che qualsiasi goal può essere descritto come massimizzazione del reward cumulativo atteso. Nel definire questa quantità, anche detta **return** si hanno due possibilità:

- Definirla come **finite-horizon undiscounted return**, ovvero fare semplicemente la somma dei reward ottenuti durante una sequenza finita di stati-azioni $\tau = (s_o, a_0, s_1, a_1, ...)$:

$$R(\tau) = \sum_{t=0}^{T} r_t$$

- Definirla come **infinite-horizon discounted return**, ovvero scontando di un fattore $\gamma \in (0, 1)$ i reward in modo che la prima

ricompensa che ottengo è quella che pesa di più, poi mano a mano che si considerano reward più lontani nel tempo essi contribuiranno sempre meno:

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$$

A questo punto è bene definire un altro elemento, ovvero il **valore** V di un certo stato s_t, questa quantità specifica qualcosa che va ben oltre il semplice reward immediato fornito dall'ambiente, essa infatti fornisce una misura di quanto è desiderabile essere in un certo stato in relazione alla sequenza di stati futuri che verranno raggiunti a partire da esso seguendo la policy attuale. Viene infatti definito come:

$$V^{\pi}(s_t) = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

Questa quantità, anche detta **Value Function** è una previsione di quanta ricompensa otterrò seguendo la policy a partire da questo stato. Il fattore γ viene introdotto sia perché ha intuitivamente senso pensare che è meglio ottenere una ricompensa adesso piuttosto che ricevere la stessa tra un certo lasso di tempo, sia perché nel caso in cui la sequenza di stati-azioni sia infinita non si convergerà mai ad un valore finito. Tanto più γ si avvicina a 0 e più l'agente diventa in un certo senso "greedy", cercando di massimizzare soltanto la ricompensa immediata, viceversa valori vicini a 1 portano l'agente a considerare sempre di più i reward lontani nel tempo.

1.2.5 *Formulazione del problema*

Come già anticipato più volte, l'obbiettivo dell'apprendimento per rinforzo è imparare una policy che massimizzi il return atteso da qualsiasi stato, a partire dalle formulazione precedenti risulta evidente che tale policy, detta ottima, è definibile come:

$$\pi^* = \arg\max_{\pi} V^{\pi}(s), \forall s \tag{2}$$

Imparare la policy ottima è l'obbiettivo, ma per raggiungerlo bisogna mettere in campo le già citate value function, che possono essere di due tipi:

- **State value function:** $V(s)$ ovvero il valore di essere in un certo stato seguendo una policy π. Definita come:

$$V^{\pi}(s) = E_{\pi}[R_t | s_t = s] \tag{3}$$

- **Action value function:** $Q(s, a)$ ovvero il valore di eseguire una certa azione a in uno stato s seguendo una certa policy. Definita come:

$$Q^{\pi}(s, a) = E_{\pi}[R_t | s_t = s, a_t = a] \tag{4}$$

Con E si intende l'Expectation, ovvero qual'è il return che ci si attende, considerando anche che nel caso di policy e/o di funzioni di transizione (tra stati) stocastiche il risultato non può essere deterministico.
Prima di derivare le equazioni fondamentali del RL può essere utile definire anche:

- **Probabilità di transizione:**

$$P^a_{ss'} = P(s_{t+1} = s' | s_t = s, a_t = a) \tag{5}$$

  Ovvero se si parte dallo stato s e si esegue l'azione a si finisce nello stato s' con probabilità $P^a_{ss'}$

- **Reward atteso:**

$$R^a_{ss'} = E(r_{t+1} | s_t = s, s_{t+1} = s', a_t = a) \tag{6}$$

  Ovvero il reward atteso che si riceve partendo dallo stato s e si esegue l'azione a finendo nello stato s'

1.2.6 *Equazione di Bellman e Q-Learning*

1.2.6.1 *Intuizione e derivazione dell'equazione di Bellman*

Al fine di semplificare la spiegazione si può prendere in considerazione il caso di policy deterministiche, tralasciando le varie probabilità introdotte.
Per ottenere una policy ottima si può sfruttare la funzione Q, essa infatti valuta la bontà dell'azione in un determinato stato, permettendo di associare un azione ad esso. Possiamo scriverla infatti come:

$$Q(s, a) = r(s, a) + \gamma V^*(\delta(s, a)) \tag{7}$$

Dove $\delta(s, a)$ è lo stato in cui finisce l'agente eseguendo l'azione a nello stato s. Q è il guadagno che riceve l'agente effettuando l'azione A dallo stato S e poi seguendo la policy ottima. Per ogni stato ed azione dovrei quindi memorizzare il rispettivo valore di $Q(s, a)$, ovviamente tale tabella sarebbe enorme, ed equivarrebbe ad a provare qualsiasi possibilità. Non si ha però ancora alcuna indicazione su come conoscere la policy ottima, è bene però notare che così com'è definita quest'ultima equazione può essere riscritta ricorsivamente, il che vedremo sarà fondamentale per poter ottenere una policy ottima:

$$Q(s, a) = r(s, a) + \gamma \max_{a'} Q(\delta(s, a), a') \tag{8}$$

Da questa equazione, detta di Bellman, è evidente come seguire la policy ottima equivale a scegliere ogni volta l'azione con il valore di Q maggiore. Al contempo, il fatto di averla definita ricorsivamente ci permette di assumere che conosceremo quel valore di Q in futuro, per poi tornare a ritroso indietro.

Quello appena evidenziato è il passaggio tipico della programmazione dinamica, si suppone infatti di conoscere la policy ottima a partire da un certo stato, anche se, in realtà, quello di cui si dispone è un approssimazione di Q, approssimazione che però alla fine tenderà al suo valore reale.
Una volta fornita una spiegazione intuitiva di come funziona la formula di Bellman, si può derivare l'equazione considerando l'action value function di 4 come base.

$$Q^{\pi}(s,a) = E_{\pi}\left[\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+1} | s_t = s, a_t = a\right]$$

$$Q^{\pi}(s,a) = E_{\pi}\left[r_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+2} | s_t = s, a_t = a\right]$$

$$Q^{\pi}(s,a) = \sum_{s'} P^{a}_{ss'}\left[R^{a}_{ss'} + \gamma E_{\pi}\left[\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+2} | s_{t+1} = s'\right]\right]$$

$$Q^{\pi}(s,a) = \sum_{s'} P^{a}_{ss'}\left[R^{a}_{ss'} + \gamma\sum_{a'} E_{\pi}\left[\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+2} | s_{t+1} = s', a_{t+1} = a'\right]\right]$$

$$Q^{\pi}(s,a) = \sum_{s'} P^{a}_{ss'}\left[R^{a}_{ss'} + \gamma\sum_{a'} \pi(s',a') Q^{\pi}(s',a')\right]$$

1.2.6.2 *L'algoritmo di Q-Learning*

Come anticipato, l'obbiettivo è apprendere la funzione Q. Essa può essere appresa applicando ricorsivamente la sua definizione grazie ad un algoritmo iterativo proposto da Watkins[50], che approssima la funzione Q ad una funzione $\hat{Q}$: L'algoritmo è molto semplice ed approssima

Algorithm 1: Algoritmo iterativo per Q-Learning

Input: gli elementi di $\hat{Q}(s,a)$;
for *m episodi* **do**
  while *episodio non terminato* **do**
    Selezionare un'azione fra quelle possibili utilizzando la tabella $\hat{Q}$ ed un criterio di scelta,ad esempio $\epsilon - greedy$;
    Eseguire l'azione, ricevere il reward r e lo stato s';
    Il nuovo valore di $\hat{Q}(s,a)$ sarà:

$$\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha[r + \gamma \max_{a'} \hat{Q}(s',a') - \hat{Q}(s,a)]$$

    Con α learning rate e γ discount factor.;
    $s \leftarrow s'$;
  end
end

direttamente la funzione Q ottimale, vanno però fatte alcune riflessioni

su di esso. Innanzitutto si può intuire che all'inizio l'agente eseguirà azioni casuali, in quanto non ha informazioni per $\hat{Q}(s, a)$, il suo obiettivo sarà quindi di esplorare lo spazio degli stati e delle azioni al fine di approssimare meglio la funzione Q, in questa fase si avrà perciò un apprendimento molto lento, essa viene detta fase di **bootstrap**.
A questo punto entra in gioco la classica questione di **exploration** vs **exploitation**, infatti quando l'agente incomincerà a disporre di informazioni su $\hat{Q}(s, a)$ potrebbe scegliere deterministicamente l'azione che lo porta ad un reward più alto, ma a quel punto potrebbe non provare ad effettuare altre sequenze di azioni che magari lo porterebbero ad una ricompensa più alta. Al contempo potrebbe scegliere sempre a caso l'azione da intraprendere, ma ciò sfocerebbe in un costo computazionale assurdo, ritardando la convergenza enormemente. Proprio per questo bisogna trovare un equilibrio fra lo sfruttamento delle informazioni di cui già si dispone ed il provare nuove soluzioni.
Una scelta tipica è quella dell'approccio $\epsilon - greedy$, sfruttando sia le informazioni a disposizione sia la naturale tendenza esplorativa delle scelte random. Infatti l'ϵ citato è la probabilità di prendere una scelta random invece di quella migliore indicata dalla funzione Q. Il principale difetto di questo approccio è che tiene comunque in considerazione principalmente solo l'azione che al momento è considerata più fruttuosa. Da non trascurare è anche la dimensione della tabella che contiene i valori di Q, la quale è grande $Q \times A$, con Q numero di possibili stati e A numero di azioni possibili. Risulta evidente come un approccio del genere non può essere adottato così com'è in problemi reali. Come spiegato nelle prossime sezioni una soluzione è quella di utilizzare una rete neuronale artificiale per approssimare la tabella contenente Q.

1.3 TASSONOMIA DEGLI ALGORITMI DI RL

Una volta introdotti intuitivamente i concetti fondamentali e comuni nel campo del Reinforcement Learning è possibile delineare una tassonomia dei principali algoritmi allo stato dell'arte della materia. Ovviamente non è banale dividere in classi ben separate approcci che talvolta comprendono molte idee e intuizioni comuni, ma la classificazione presentata può comunque aiutare a capire le motivazioni che stanno dietro la scelta di un certo algoritmo quando si deve affrontare un problema con il RL. Come si indicato nella figura 2, esistono due famiglie principali di algoritmi di reinforcement learning, ovvero gli algoritmi che hanno un modello dell'ambiente (fornito o imparato), e quelli che non lo hanno, rispettivamente **Model-based** e **Model-free**.
Un algoritmo model-based dispone quindi della capacità di prevedere transizioni di stato e rispettivi reward. Risulta evidente come questo approccio, oltre ad avere ovvi parallelismi con il funzionamento della mente umana, può essere molto efficace perché permette di pianificare strategie per raggiungere determinati obiettivi. Oltre a questo già im-

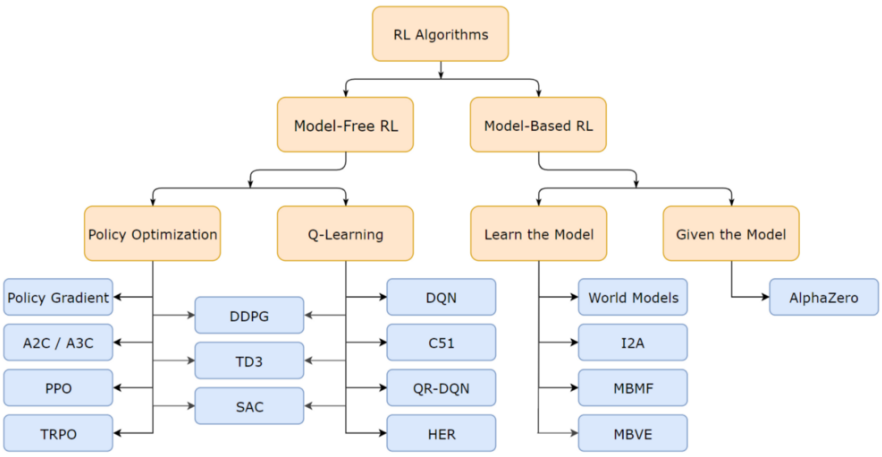


Figura 2: Tassonomia degli algoritmi di Reinforcement Learning,da OpenAI [30]

portante aspetto, gli algoritmi model-based possono sfruttare questo modello interno dell'environment per il proprio addestramento, evitando di utilizzare l'ambiente vero e proprio.
Nonostante questi aspetti molto interessanti, gli algoritmi della famiglia Model-Free sono però i più famosi ed utilizzati[30]. Il motivo per cui, molto spesso, non viene sfruttato un modello dell'ambiente è che, a parte qualche caso particolare come quello di AlphaZero[39], non se ne ha a disposizione uno. Tale limitazione obbliga l'agente a desumere questo modello dalla sua esperienza in fase di addestramento, il che rappresenta una sfida non banale. Sebbene esistano anche in questo caso degli esempi notevoli, come il lavoro di Ha et Schmidhuber[11] nel quale il modello del mondo appreso viene sfruttato per addestrare l'agente, questi algoritmi sono particolarmente sensibili all'overfitting, in quanto potrebbero trovare dei modi per migliorare la propria prestazione nell'ambiente simulato che non sono sfruttabili in quello reale. Sopratutto nel caso di ambienti complessi è davvero difficile creare una rappresentazione fedele del mondo, richiedendo perciò un costo computazionale a volte proibitivo.
Gli algoritmi model-free si sono dimostrati più facili da implementare e da adattare a vari ambiti di utilizzo, pur richiedendo, ovviamente, un maggior numero di tentativi per imparare correttamente ad eseguire un task. L'ammontare di esperienza che l'agente deve accumulare per ottenere un certo livello di performance è detta **Sample-Efficiency** ed è una buona misura di comparazione tra algoritmi di RL, come mostrato in figura 3.
Ovviamente quello appena presentata è solo il primo grado di suddivisione degli algoritmi di RL, nelle sezioni successive verranno mostrate le ulteriori sottoclassi presenti nelle due principali famiglie nel RL. Divisione che si posa su una domanda fondamentale, ovvero che cosa l'algoritmo deve imparare.

Figura 3: Confronto della Sample-Efficiency fra i vari approcci del RL, tratta da [9]

1.4 MODEL FREE RL

Come anticipato l'ambito degli algoritmi Model-Free è molto ampio ed è teatro di un fervente lavoro di ricerca, sopratutto negli ultimi anni, anche grazie ad un sempre crescente interesse verso il reinforcement-learning.
Come evidenziato in figura 2, in questo ambito possiamo distinguere tra gli algoritmi legati al Q-Learning e quelli legati alla Policy Optimization. Queste due possibili vie rappresentano ciascuna una possibile risposta alla domanda posta già in precedenza, ovvero cosa deve imparare l'algoritmo durante il suo addestramento?

- Gli algoritmi che si ispirano al **Q-Learning** cercano di imparare una funzione $\hat{Q}(s, a)$ che approssimi la funzione action-value ottimale $Q^*(s, a)$. L'obbiettivo è quindi di stimare il valore di un azione in un certo stato, per poi selezionare quella migliore in base a queste stime, tipicamente utilizzando come obiettivo la funzione di Bellman, come spiegato nella sezione 1.2.6.2. Il raffinamento della funzione $\hat{Q}(s, a)$ è fatto **off-policy**, ovvero che l'aggiornamento può utilizzare i dati raccolti in qualsiasi momento durante l'addestramento, indipendentemente da come l'agente ha scelto di esplorare l'ambiente quando i dati sono stati ottenuti.

- I cosiddetti **Policy Gradient Algorithms** invece manipolano direttamente la **policy**, che in questo caso è parametrizzata. Essa viene formalizzata come $\pi_\theta(a|s)$ con θ che rappresenta i vari parametri da ottimizzare e non necessita di una Value function esplicita. Solitamente l'ottimizzazione avviene **On-Policy**, ovvero ogni aggiornamento dei parametri viene eseguito utilizzando le esperienze ottenute seguendo l'ultima versione della policy.

A titolo esemplificativo verranno presentati alcuni esempi di algoritmi per entrambe le famiglie di algoritmi Model-Free ed alcuni che prendono ispirazione da entrambi gli approcci.

1.4.1 *Metodi ispirati al Q-Learning*

Nelle sezioni precedenti si è già parlato del principi fondanti del Q-Learning, che in estrema sintesi mira ad approssimare la funzione Q ottimale, ovvero un valore per ogni coppia stato azione. Risulta evidente

come tale approccio sia computazionalmente insostenibile quando lo spazio degli stati e delle azioni è molto grande.
Una possibile soluzione a questo problema è quello di approssimare i valori di Q con una funzione parametrica del tipo $Q(s, a; \theta)$, purtroppo la letteratura [43, 42] ha evidenziato più volte come difficilmente si possa arrivare alla convergenza tramite questo semplice accorgimento.

1.4.1.1 *DQN*

Anche per superare questi problemi nel 2013 è stato presentato il modello DQN[23], questo lavoro, poi raffinato ed esteso con l'articolo Human-level control through deep reinforcement learning [24] ha avuto un effetto dirompente sull'intero campo del Reinforcement Learning.
Nel lavoro citato gli autori hanno proposto un modello che si è dimostrato capaci di imparare a giocare a 50 diversi giochi per la console Atari 2600 solo guardando le immagini RGB provenienti dal gioco e la variazione del punteggio in relazione alle proprie mosse.
Come anticipato, rappresentare la funzione Q con un approssimatore di funzioni parametrico (ad esempio una rete neurale), porta a risultati non soddisfacenti, sopratutto a causa del fatto che la funzione obiettivo Q viene aggiornata continuamente, non si ha un feedback diretto tra le azione intraprese con una policy e la versione di Q utilizzata per desumerla e piccoli cambiamenti in Q possono provocare grandi cambiamenti nella policy.
Per superare queste limitazioni ed ottenere risultati soddisfacenti i ricercatori di Deep Mind hanno messo in campo diverse soluzioni, tra le principali:

- Utilizzo di una **Convolutional Neural Network**: considerato che l'algoritmo deve riuscire a capire come giocare guardando sequenze di immagini RGB, si è rivelato particolarmente vantaggioso utilizzare delle reti con degli strati di convoluzione, come evidenziato in figura 4, in modo da considerare regioni di immagini e non ogni pixel indipendentemente. Tali architetture permettono solitamente di mantenere le relazioni spaziale fra oggetti nell'immagine e di astrarre informazioni di più alto livello.

- **Experience Replay**: questo meccanismo, biologicamente ispirato[20, 31], si basa sull'idea che immagazzinando le esperienze dell'agente in un apposito buffer dal quale poi estrarre successivamente dei batch random per garantire un apprendimento più robusto. Questo approccio impedisce che la rete venga addestrata solo in base alle esperienze appena compiute nell'ambiente, permettendo perciò di essere meno sensibili ai piccoli cambiamenti nella distribuzione dei dati che possono verificarsi. Ovviamente trattandosi di un buffer le esperienze più vecchie vengono mano a mano scar-

tate a favore di quelle nuove, permettendo di avere comunque un training coerente.

- **Rete separata per la funzione Q obiettivo**: molto brillantemente, gli autori hanno proposto di utilizzare una seconda rete neuronale durante l'addestramento. Essa viene utilizzata per generare i valori target di Q, i quali vengono poi sfruttati per calcolare la funzione di costo. Questo approccio va ad evitare quella situazione pericolosa in cui ad ogni passo di addestramento i valori di Q vengono cambiati, ma vengono utilizzati sia per decidere quali azioni intraprendere sia per per approssimare meglio Q, portando ad instabilità gravi nell'addestramento. Al contrario in questo caso i valori di Q-Target vengono mantenuti stabili e solo periodicamente vengono aggiornati.

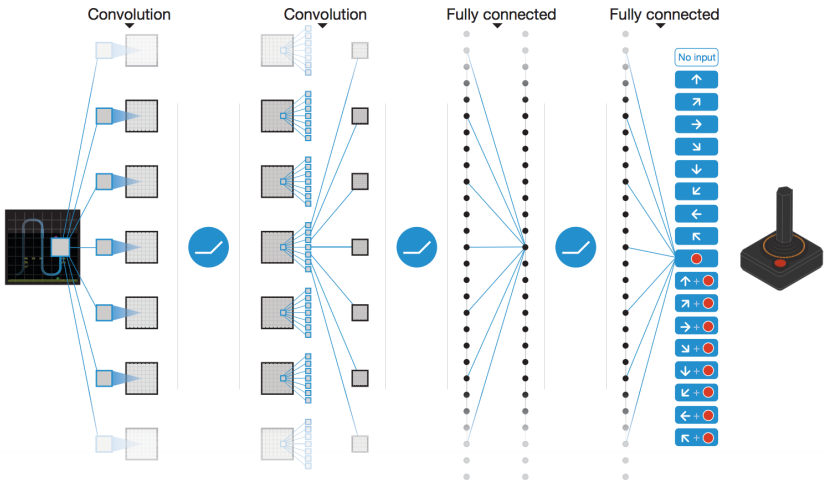


Figura 4: Schema sommario dell'architettura di una DQN, tratta da [23]

A partire da questo lavoro iniziale negli anni sono stati poi sviluppati diversi nuovi accorgimenti per migliorare ulteriormente le prestazioni di questi modelli. Assolutamente degni di nota sono il principio del **Prioritized Experience Replay**[36] e del **Dueling DQN**[48].
Il primo meccanismo permette di dare maggiore importanza alle esperienze nel buffer su cui sono stati fatti errori più grandi nella stima del Q-value, aumentando la probabilità che esse vengano incluse tra quelle utilizzate ad ogni step di training.

Per spiegare il secondo meccanismo bisogna però introdurre brevemente il concetto di **Advantage function**. Questa quantità può essere pensata come un indice di quanto è preferibile eseguire una certa azione rispetto alle altre. Il valore di una certa azione in un certo stato può essere quindi riscritto come:

$$Q(s,a) = V(s) + A(a)$$

In una Dueling DQN si ha appunto una rete che calcola separatamente A e V, per poi combinarle sono all'ultimo livello di essa. L'intuizione

di base è che è preferibile conoscere quali stati sono veramente di valore, senza dover conoscere l'effetto di ogni azione in ogni stato. Tale accorgimento è particolarmente utile sopratutto in quegli stati dove a prescindere dall'azione che si esegue non si hanno particolari risultati.

1.4.2 *Metodi policy based*

I metodi finora descritti avevano come obbiettivo quello di apprendere una value function su stati/azioni per poi fondare la propria policy su queste stime. In questo caso si vuole invece modellare ed ottimizzare direttamente la policy, solitamente descritta come funzione parametrizzata rispetto a θ. L'assunzione è che il reward guadagnato dipenda dalla policy, perciò i suoi parametri θ devono essere ottimizzati per ottenere il maggior reward.
La policy è definibile quindi come $\pi_\theta(a|s) = P[a|s]$, ovvero come probabilità di eseguire l'azione a nello stato s avendo parametri θ. Ricordando che l'obiettivo è quello di ottimizzare una funzione di score $J(\theta) = E_{\pi\theta}[\sum \gamma r]$, bisogna prima quantificare la **qualità della policy**, per poi usare **salita del gradiente** per trovare i parametri θ atti a migliorare la policy θ.
Bontà della policy, $J(\theta)$: in uno spazio di stati continuo questa quantità può essere descrita come:

$$J(\theta) = E_\pi = \sum_s d(s) \sum_a \pi\theta(s,a) R_s^a \tag{9}$$

Dove $d(s)$ è la probabilità di essere nello stato s.
$\pi\theta(s,a)$ probabilità di eseguire l'azione a nello stato s utilizzando la policy π.
R_s^a è il reward immediato ottenuto.
Salita del gradiente per la policy: Avendo a disposizione una funzione parametrica che indica la bontà della policy in base ai reward ottenuti si possono trovare i parametri migliori per massimizzare il reward tramite ascesa del gradiente. In questo caso non si parla, come spesso accade, di discesa del gradiente perché non stiamo considerando una funzione di costo bensì una di score.
Riassumendo l'idea è quindi quella di avere una policy π_θ, una funzione obiettivo $J(\theta)$, un gradiente $\nabla_\theta J(\theta)$ ed aggiornare i parametri con $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$
Per arrivare a questo risultato bisognerà trovare i parametri ottimali θ^* che massimizzano lo score, ovvero:

$$\theta^* = \arg\max_\theta E_{\pi\theta}[\sum_t R(s_t, a_t)] \tag{10}$$

Prima di mostrare come si può effettuare la salita del gradiente è bene soffermarsi sull'equazione 9, come si può notare i parametri θ cambiano come le azioni vengono scelte e di conseguenza, quali reward verranno ricevuti e quali stati verranno visitati. Non è però semplice capire

come migliorare la performance della policy, in quanto essa dipende sia dalle azioni scelte che dalla distribuzione degli stati in cui esse vengono compite. Il problema è che non si conosce l'effetto della policy sulla distribuzione degli stati.
Proprio per superare questa problematica si può utilizzare il cosiddetto **"Policy Gradient Theorem"**, attraverso il quale si può calcolare il gradiente ma senza derivare rispetto alla distribuzione degli stati. Senza scendere nella complessa dimostrazione esso prescrive che per qualsiasi policy differenziabile $\pi_\theta(s, a)$ il suo gradiente é:

$$\nabla_\theta J(\theta) = E_{\pi\theta}[\nabla\theta \log \pi_\theta(s, a) Q^{\pi_\theta}(s, a)] \tag{11}$$

1.4.3 *Metodi ibridi*

Dopo aver analizzato entrambe le famiglie di algoritmi di Model-Free RL si possono trarre alcune conclusioni:

- Il vantaggio principale dei metodi **policy-based** è che sono orientati direttamente ad ottimizzare quello che ci interessa davvero, ovvero la policy, motivo per cui essi tendono ad essere stabili ed affidabili. Essi sono inoltre preferibili quando si trattano ambienti stocastici e/o con spazio degli stati continuo.

- Al contrario i metodi di Q-learning ottimizzano solo indirettamente la policy, in quanto il loro obbiettivo è approssimare al meglio la funzione Q_θ. Proprio per questo tendono ad essere meno stabili, tuttavia nel caso in cui non incorrano in problemi essi dimostrano una maggior sample-efficiency.

E' evidente quindi come ogni approccio abbia i suoi pregi e difetti, motivo per cui la ricerca si è interrogata se si potessero combinare insieme per sfruttare i pregi di entrambi. Uno dei tentativi, più famoso e di maggior successo, di rispondere a questa domanda è l'approccio Actor-Critic.

1.4.3.1 *Approccio Actor-Critic*

L'idea fondamentale di questo metodo è di separare il modello in due parti:

- L'**Actor** che aggiorna i parametri θ della policy $\pi_\theta(a|s)$ e perciò decide quale azione intraprendere. Essenzialmente il suo obbiettivo è di imparare una policy ottima per far muovere l'agente. E' evidente come questa parte sia ispirata ai metodi policy-based.

- Il **Critic** giudica le azione dell'actor, aggiornando costantemente i parametri w della value function, la quale può essere o del tipo action-value $q_w(a|s)$ o del tipo state value $V_w(s)$.

Grazie alle indicazione del critico l'actor potrà quindi aggiustare la policy in modo molto più efficace rispetto ai metodi tradizionali che non fanno uso di questo accorgimento. L'idea che sta alla base di questo approccio è davvero intuitiva, basandosi sul fatto che abbiamo due componenti separata (ma interagenti) che miglioreranno in modo incrementale la loro capacità, rispettivamente, di decidere cosa fare e di valutare quanto vantaggiosa è stata la scelta intrapresa.

1.4.3.2 *Asynchronous Advantage Actor-Critic (A3C)*

L'algoritmo A3C[25], presentato dai ricercatori di DeepMind nel 2016 si basa proprio su questo principio della divisione fra Actor e Critic. Deve il suo nome a tre degli aspetti principali che lo descrivono:

- **Asincrono**: al contrario di molti approcci dove un singolo agente interagisce con l'ambiente, in questo caso si sfruttano una molteplicità di agenti che interagiscono ognuna con una copia differente dell'environment. Ogni agente ha i suoi parametri della rete, distinti dagli altri, perciò, oltre ad avere l'ovvio aumento delle prestazioni dovute al parallelismo, si riesce ad ottenere un insieme di esperienze diverse ed indipendenti tra loro, aiutando l'addestramento. Questa struttura è ben evidenziata in figura 5. Il termine asincrono è dovuto al fatto che l'aggiornamento della rete globale non avviene simultaneamente.

- **Actor-Critic**: rispetta le caratteristiche dei metodi introdotti nella sezione 1.4.3.1.

- **Advantage**: similmente a quanto avviene nelle Dueling DQN(introdotte nella sezione 1.4.1.1),nella regola di aggiornamento invece di utilizzare il Discounted-Reward come metrica di valutazione dello stato, viene utilizzato il concetto di advantage. Il vantaggio di utilizzare $A = R - V(s)$, permette di capire quanto migliore rispetto alle attese si è rivelato intraprendere una certa azione. In questo caso si utilizza R come stima di $Q(s, a)$

L'algoritmo prevede che gli agenti vengano addestrati in parallelo e periodicamente la rete globale viene aggiornata. Come già introdotto, dopo che un agente ha aggiornato i propri parametri li invia alla nodo centrale, dal quale riceve i nuovi valori di essi, per poi tornare all'addestramento nel suo specifico ambiente per un certo numero di passi. Risulta evidente quindi come ogni agente possa sfruttare anche le esperienze fatte da gli altri, aumentando la varietà del suo training set.

1.4.3.3 *Advantage Actor-Critic (A2C)*

Questo algoritmo, come intuibile, è la versione sincrona di A3C. Il seppur ottimo A3C, soffriva del fatto che, essendo gli aggiornamenti

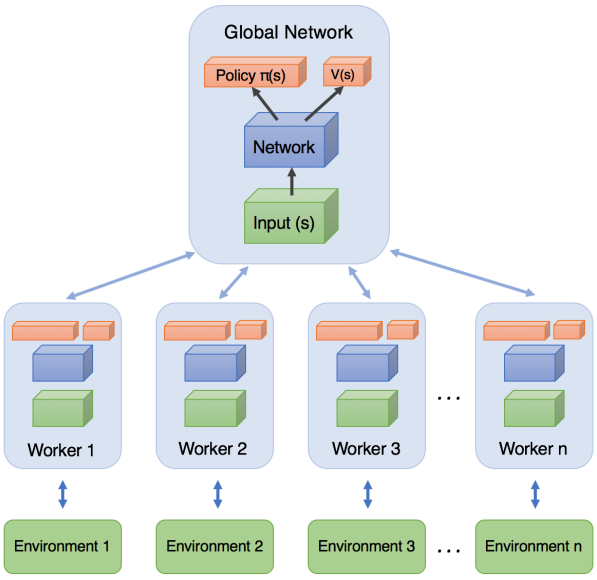


Figura 5: Schema dell'architettura generale di A3C. Tratta da [25].

della rete globale indipendenti tra loro, certi agenti potevano raccogliere esperienze seguendo diverse versioni della policy, portando a degli aggiornamenti non ottimali. Proprio per risolvere questa limitazione, in A2C esiste un coordinatore che aspetta che tutti gli agenti finiscano il proprio lavoro prima di aggiornare la rete globale, così da permettere poi agli agenti di partire tutti con la stessa policy, ma ovviamente in ambienti diversi.
Questa implementazione si è dimostrata migliore della versione asincrona, dimostrando anche di utilizzare le GPU in modo più efficiente.[1]
Va però considerato che di fatto con questo accorgimento è poco utile avere un set di parametri separati per ogni agente, perché di fatto essendo sincroni l'aggiornamento è come avere n copie dello stesso agente. L'unica cosa che varia sono gli ambienti in cui vengono addestrati.

1.5 MODEL-BASED RL

Un campo molto affascinante nell'ambito del reinforcement learning è senz'altro quello degli algoritmi model-based, essi, a differenza dei model-free, cercano di desumere un modello dell'ambiente, in modo da prevedere distribuzioni degli stati, dei reward e delle transizioni. La loro componente principale è quindi quella della **pianificazione**, mentre i model-free si basano principalmente sull'apprendimento[42]. Essi infatti disponendo di un modello dell'ambiente (fornito o appreso) possono simulare un intero episodio (o un modello di distribuzione dei possibili episodi), compiendo il proprio addestramento in un ambiente simulato.

1 https://openai.com/blog/baselines-acktr-a2c/

In questo caso è più difficile fare una distinzione netta fra famiglie di algoritmi, motivo per cui l'unico criterio per poterli dividere è se hanno a disposizione un modello dell'ambiente oppure lo devono apprendere.

1.5.1 *Modello da apprendere*

Nel caso in cui non si ha a disposizione un modello dell'ambiente può essere comunque utile tentare di apprenderlo in base alle esperienze raccolte dall'agente. Una volta che il modello appreso si dimostra sufficientemente accurato lo si può sfruttare per ottenere delle esperienze aggiuntive sul quale addestrare l'agente. A questo proposito possiamo distinguere due casi:

- Le esperienze tratte dal modello dell'ambiente vengono utilizzate in aggiunta a quelle proveniente dall'ambiente "reale". Un caso tipico è di MBVE[8].

- L'addestramento viene compiuto completamente sulla base delle esperienze raccolte dal modello dell'ambiente appreso, ovvero il caso di World Model[11].

Il secondo caso è, a mio parere, molto interessante e merita un breve approfondimento.
Come introdotto, World Models è un approccio che si prefigge di addestrare l'agente in una versione appresa e totalmente sintetica dell'ambiente originale. Il rischio di questa metodologia è però quella di avere un comportamento risultante ottimale per l'ambiente sintetico ma non soddisfacente per quello originale, in quanto la policy potrebbe sfruttare caratteristiche uniche del modello appreso, non ritrovabili nell'originale. Al fine di superare questa problematica, gli autori introducono un parametro τ (temperature) per controllare l'ammontare di incertezza dell'ambiente generato, in modo da avere maggior variabilità durante l'addestramento.
In figura 6(tratta da [2] viene ben descritto il principio di funzionamento di questo approccio, di cui i componente principali sono:

- **VAE(V)**: al fine di ottenere una rappresentazione compressa dell'immagine di partenza viene utilizzato un Variational Autoencoder,. il quale produce un vettore latente z_t, dal quale si può ricostruire l'originale, ovviamente con delle perdite dovute alla compressione.

- **MDN-RNN(M)**: mentre l'obiettivo di V era quello di comprimere cosa l'agente vede ad ogni frame, l'obiettivo di M è di comprimere cosa avviene nel tempo, per cui il suo ruolo è quelli di prevedere lo stato futuro. Data la natura stocastica di molti

2 https://worldmodels.github.io/

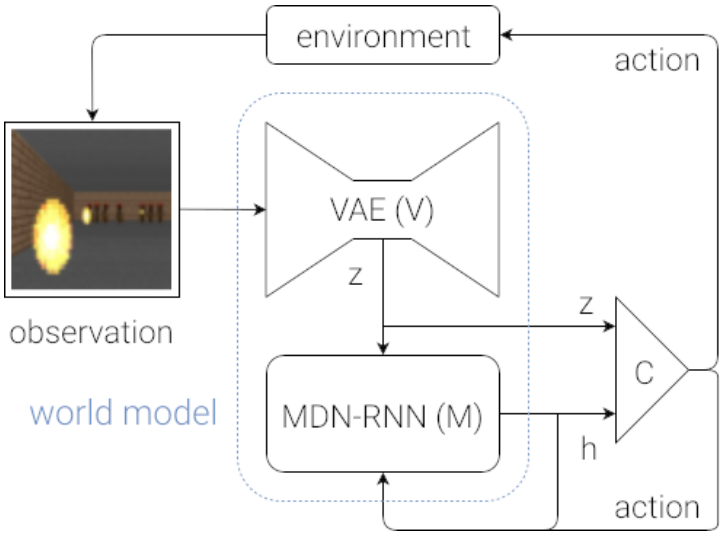


Figura 6: Schema del funzionamento di World Models. Tratta da [11]

ambienti come output di M non si ha una predizione deterministica di z_{t+1} bensì una funzione della densità di probabilità $p(z)$. $p(z)$ viene approssimata come mix di distribuzioni Gaussiane, addestrando la RNN a a modellarla come $P(z_{t+1}|a_t, z_t, h_t)$ dove a_t è l'azione eseguita al tempo t e h_t è lo stato nascosto della RNN al temp t. Come anticipato in questo caso si può utilizzare un parametro τ per controllare il grado di variabilità del modello.

- **Controller(C)** questa porzione del modello è direttamente responsabile della scelta dell'azione da eseguire per massimizzare il reward a partire dall'input ricevuto da V e M . Nel lavoro presentato gli autori propongono di usare una rete il più semplice possibile, ad esempio una rete feed-forward con un solo layer. La motivazione per questa scelta è che la complessità dell'agente risiede completamente in V e M.

1.5.2 *Modello fornito*

In questo caso il modello dell'ambiente viene fornito, quindi che esso sia un insieme di leggi fisiche, delle regole di un gioco o qualsiasi altro tipo di formulazione deterministica. Un esempio classico di algoritmo model-based con modello fornito è Alpha-Zero [39].
Trattandosi di un algoritmo che deve agire in un gioco discreto, deterministico e con completa osservabilità dello stato, si fa uso della tecnica Monte Carlo tree search, il che rappresenta una scelta naturale per questo tipo di task, in quanto sopratutto in giochi complessi come gli scacchi o il Go è impossibile esplorare tutto l'albero dei possibili stati. Differentemente da approcci precedenti, AlphaZero non viene preaddestrato con learning supervisionato, bensì impara a giocare sfidando se stesso. Deve il suo nome proprio al fatto che il suo addestramento

parte da zero, senza nessuna conoscenza su come giocare a Go, fuorché, ovviamente, le regole del gioco stesse. Il modello è quindi composto da due reti separate, le quali cercano di migliorare la propria policy e valutare il valore degli stati, cercando di prevedere chi vincerà la partita. L'addestramento viene eseguito a partire dalle esperienza ricavate eseguendo ricerche Monte Carlo, portando ad un progressivo miglioramento della policy.

2

PROXIMAL POLICY OPTIMIZATION

Tra gli algoritmi di reinforcement learning allo stato dell'arte non si può non citare PPO, sigla che sta per Proximal Policy Optimization[38]. Esso si è dimostrato particolarmente valido e generale, permettendone la sua applicazione a vari e diversi campi, dimostrando di poter addirittura essere utilizzato per giocare a Dota 2 [29], un e-sport fra i più famosi al mondo, riuscendo nel 2019 nel compiere l'impressionante impresa di sconfiggere i campioni mondiali del gioco in diretta streaming.
Prima di spiegare nel dettaglio il funzionamento dell'algoritmo verranno passati in rassegna gli algoritmi con i quali condivide alcuni aspetti principali.
A titolo esemplificativo verranno infine presentati alcune applicazione di PPO a task anche complessi.

2.1 BACKGROUND

Al fine di comprendere al meglio i principi e le motivazioni delle scelte adottare nel progettare PPO, è bene partire da due suoi algoritmi precursori, ovvero Vanilla Policy Gradient e Trust Region Policy Optimization.

2.1.1 *Vanilla Policy Gradient*

Come intuibile dalla sezione 1.4.2, l'obiettivo principale degli algoritmi policy based è quello di abbassare la probabilità di scegliere delle azioni che portano a basse ricompense ed alzare quella di eseguire azioni che portano a reward più alti.
Vanilla Policy Gradient è un implementazione dei metodi policy-gradient, esso basa le sue stime di return atteso ipotizzando di seguire la policy attuale, si tratta perciò di un algoritmo **On-Policy**.
L'algoritmo basa il suo funzionamento sull'aggiornamento dei parametri della policy attraverso salita del gradiente stocastica:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_{\theta_k})$$

Dato $J(\pi_\theta)$ come finite-horizon undiscounted return atteso della policy, il suo gradiente è:

$$\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) A^{\pi_\theta}(s_t, a_t)\right]$$

ove τ è la sequenza di stati-azioni e A^{π_θ} l'advantage function della policy attuale.
Di seguito viene riportato l'algoritmo in pseudo-codice, tipicamente, quando vengono selezionate le azioni da eseguire si ha una cerca componente casuale, in modo da incoraggiare l'agente ad esplorare lo spazio delle soluzioni, riducendo poi questa componente via via che l'addestramento procede, in modo da sfruttare le conoscenze già acquisite. Va però considerato che ridurre la casualità potrebbe portare ad incorre in soluzione sub-ottimali.

Algorithm 2: Algoritmo Vanilla Policy Gradient

Input: parametri iniziali θ_o,ϕ_0 di policy e value function;
for $k = 0, 1, 2, ...$ **do**
Collezionare un insieme di traiettorie $D_k = \tau_i$ seguendo la policy $\pi_k = \pi(\theta_k)$;
Calcolare i reward $\hat{R}_t$;
Calcolare $\hat{A}_t$ sulla base della di $V_{\varnothing_k}$;
Stimare il gradiente della policy come:

$$\hat{g_k} = \frac{1}{D_k} \sum_{r\in D_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \hat{A}_t$$

Calcolare l'aggiornamento della policy come:

$$\theta_{k+1} = \theta_k + \alpha \hat{g}_k$$

Aggiustare la value function tramite regressione o MSE con un algoritmo di discesa del gradiente:

$$\theta_{k+1} = \arg\min_\phi \frac{1}{|D_k|T} \sum_{r\in D_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t)\right)^2$$

end

2.1.2 *Trust Region Policy Optimization*

I cosidetti metodi policy gradient, pur essendo efficaci per spazi di stati ed azioni continui e molto ampi, soffrono di alcune limitazioni principali. Oltre ai classici problemi, quali grande variabilità nella valutazione della policy (sopratutto se non si usano modelli actor-critic) e bassa sample-efficiency, essi sono fortemente influenzati sia dal numero di step

da eseguire prima di aggiornare la policy e sia dal learning rate.
Infatti, se si utilizzano pochi step l'addestramento convergerà troppo lentamente, mentre se se ne considerano troppi la policy diventerà completamente instabile. Il problema è che determinare un valore consono per questi parametri non è affatto semplice e va oltre il semplice tuning degli iper-parametri. Quello che servirebbe è un metodo più accurato che ci assicuri che qualsiasi cambiamento della policy garantirà un miglioramento della performance.
Proprio per rispondere a queste esigenze nasce TRPO[37], che si pone proprio l'obbiettivo di assicurare che la policy non cambi troppo tra un update e l'altro, aggiungendo quindi un vincolo che forzi la permanenza della policy in una cosiddetta **trust region**.
Questo vincolo è espresso in termini di KL-Divergence, ovvero una sorta di misura di distanza fra due distribuzioni di probabilità.
Il vincolo posto è quindi che la KL-Divergence tra nuova e la vecchia policy sia minore di δ, dimensione della regione. Formalizzando l'aggiornamento dei parametri θ della policy π consiste in :

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}(\theta_k, \theta)$$
$$con\ \hat{D}_{KL}(\theta||\theta_k) \leq \delta$$

Dove $\mathcal{L}(\theta_k, \theta)$ è detto surrogate advantage, ovvero una misura di come la policy π_θ si comporta rispetto alla vecchia policy π_{θ_k} usando i dati provenienti dalla vecchia policy:

$$\mathcal{L}(\theta_k, \theta) = \underset{s,a\sim\pi_{\theta_k}}{E}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a)\right]$$

Infine $\hat{D}_{KL}(\theta||\theta_k)$ è la KL-divergence media tra le policy sugli stati visitati dalla policy precedente:

$$\hat{D}_{KL}(\theta||\theta_k) = \underset{s\sim\pi_{\theta_k}}{E}\left[D_{KL}(\pi_\theta(\cdot|s)||\pi_{\theta_k}(\cdot|s))\right]$$

Teoreticamente queste sono le formule sul quale l'algoritmo si basa, nella pratica però bisogna prendere diversi accorgimenti per migliorare la loro trattabilità, perché così come poste è troppo poco efficiente tramite discesa del gradiente naturale. Senza scendere troppo nei complessi dettagli matematici l'algoritmo si può riassumere in questi passi:

- Si raccolgono un set di esperienze secondo la policy attuale
- Si stima l'advantage $\hat{A}_t$ sulla base value function corrente.
- Si stima il gradiente della policy.
- Si utilizza il gradiente coniugato per calcolare l'Hessiana della KL-divergence.
- Si aggiorna la policy in modo da migliorare il Surrogate Advantage ma rispettando i vincolo della KL-Divergence.

- Si aggiorna la value function

Come è intuibile dalla spiegazione appena fornita, TRPO è un algoritmo molto complesso e potente, che si è dimostrato efficace in molti task. Nonostante ciò, il vincolo imposto risulta troppo computazionalmente oneroso da calcolare e ciò rende questo approccio difficilmente sfruttabili in caso di modelli di dimensioni notevoli.

2.2 L'ALGORITMO

In questa sezione verrà spiegato nel dettaglio PPO, un algoritmo di RL proposto da Schuman et al.[38] nel 2017, esso rappresenta un grande traguardo, in quanto si è dimostrato performante quanto o più degli altri algoritmi allo stato dell'arte, riuscendo, al contempo, ad essere molto più semplice da implementare e da applicare[28].

2.2.1 *Idea di base*

Proximal Policy Optimization nasce essenzialmente per rispondere alla stessa esigenza che cerca di soddisfare l'algoritmo TRPO spiegato alla sezione 2.1.2, ovvero evitare degli aggiornamenti troppo grandi della policy ma senza rendere l'addestramento esasperatamente lento. TRPO, pur dimostrandosi affidabile e capace di soddisfare questa esigenza, è considerato troppo complicato, principalmente a causa del fatto che utilizzare il vincolo sulla KL-Divergence fuori dalla funzione obiettivo lo rende particolarmente oneroso computazionalmente.
L'intuizione su cui si basa PPO è quella di includere il vincolo direttamente nella funzione obiettivo, utilizzando solamente ottimizzazioni di primo ordine. Come sarà evidente durante la spiegazione dell'algoritmo, esso si dimostra molto più semplice ed intuitivo, pur mantenendo performance allo stato dell'arte.
Come in TRPO l'obbiettivo dell'addestramento è massimizzare una "funzione obiettivo surrogata" che fornisce una stima di quanto la performance della nuova policy cambierà dopo il suo aggiornamento. Questa funzione obiettivo viene utilizzata al posto della classica utilizzata in nell'algoritmo Vanilla Policy Gradients:

$$J(\theta) = E_t[\log \pi_\theta(a_t|s_t)\hat{A}_t] \tag{12}$$

Il concetto di base è quindi che i parametri θ della policy verranno aggiustati tramite ascesa del gradiente in modo da incentivare le azioni più vantaggiose, l'impatto dell'azione viene quindi descritto dalla quantità $\log \pi_\theta(a|s)$. Un modo alternativo per descriverlo è quello di utilizzare la probabilità di un azione secondo la policy corrente dividendo questa quantità per la probabilità della stessa azione seguendo la

vecchia policy[17], questa quantità, detta rapporto di probabilità $r_t(\theta)$ è descritta come:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)},\ perciò\ r(\theta_{old}) = 1 \tag{13}$$

Dall'equazione 13 risulta evidente che :

- Se $r_t(\theta) > 1$: l'azione è più probabile nella policy corrente rispetto a quella precedente.
- Se $0 < r_t(\theta) < 1$: l'azione è meno probabile nella policy corrente rispetto a quella precedente.

Riprendendo la funzione obiettivo 12 sostituiamo il termine $\log \pi_\theta(a|s)$, come accade in TRPO:

$$L(\theta) = \hat{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t\right] \tag{14}$$

Il problema di questa formula nasce dal fatto che se l'azione diventa molto molto più probabile ciò porterà ad un gradiente molto ripido, portando ad una policy molto diversa dall'attuale, potenzialmente rendendo instabile l'addestramento. Nella sezione 2.1.2 è stato descritto sommariamente come TRPO cerca di superare questo problema, ovvero ponendo dei vincoli alla KL-Divergence, con tutti i calcoli e le approssimazioni necessarie per rendere implementabile la soluzione.

2.2.2 *L'aggiornamento della policy*

2.2.2.1 *L'obbiettivo dell'ottimizzazione*

La funzione obbiettivo che viene proposta dagli autori per rispondere alle esigenze già descritte, ma attraverso un approccio più semplice ed intuitivo, è quindi la seguente:

$$L^{CLIP}(\theta) = \hat{E}_t\left[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right] \tag{15}$$

Con $r_t(\theta)$ definito come nell'equazione 13 e ϵ è un iper-parametro, tipicamente uguale a 0.2. Quindi di fatto il primo termine che appare nella funzione di minimo è lo stesso dell'equazione 14, mentre il secondo modifica l'obiettivo surrogato costringendolo a rimanere nell'intervallo $[1-\epsilon, 1+\epsilon]$, rimuovendo perciò l'incentivo di avere $r_t(\theta)$ fuori dall'intervallo.
Come evidenziato dalla formula viene poi scelto il minore fra i due termini, è bene però spiegare in dettaglio quanto espresso dalla formula. Per farlo viene comodo utilizzare il grafico in figura 7, nella parte sinistra di, è rappresentata la situazione in cui $A > 0$, ovvero se l'azione presa ha avuto un buon effetto, mentre nella parte destra la situazione $A < 0$, ovvero se l'azione ha avuto un effetto negativo, per cui:

- Nel primo caso il valore di $r_t(\theta)$ viene tagliato ad $1+\epsilon$ se è troppo alto a seguito di un aumento di probabilità dell'azione troppo grande nella nuova policy rispetto alla vecchia. In queste situazione si vuole infatti evitare di essere troppo "greedy", in quanto non può essere certo un solo passo di update a determinare un cambiamento così grande. Intuitivamente si può riassumere che se l'azione era buona ed è diventata ancora più probabile seguendo il gradiente è meglio non continuare troppo ad aggiornarla in quella direzione. Se invece diventa meno probabile non c'è nessuna limitazione a di fatto annullare lo step precedente.

- Nel secondo caso il taglio viene effettuato vicino allo 0 , dove l'azione secondo la policy è poco probabile. Questo evita di rendere ancora meno probabile un azione che è già stata resa poco probabile. Intuitivamente si può riassumere che se un'azione non è buona e diventa sempre meno probabile è meglio non diminuire troppo la sua probabilità, ma se invece diventa più probabile non c'è problema nel seguire il gradiente verso la direzione opposta.

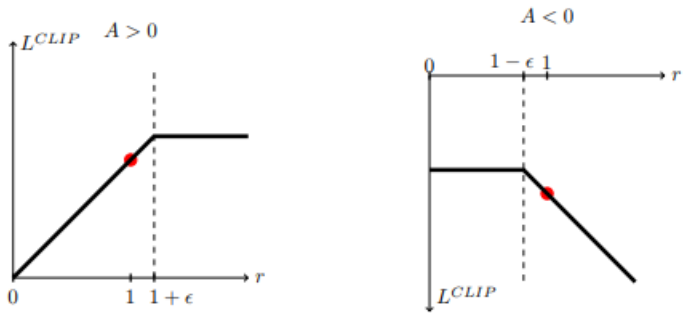


Figura 7: Il grafico mostra un singolo update della funzione L_{CLIP} in funzione del rapporto di probabilità r per Advantage positivi a sinistra e negativi a destra. Il puntino rosso indica il punto di partenza dell'ottimizzazione, ovvero r = 1. Tratta da [38]

La scelta di prendere il minimo tra i due valori si può spiegare intuitivamente con la volontà di fornire la possibilità di riparare ai possibili passi falsi nell'update della policy, infatti grazie alla presenza di $r_t(\theta)$ puro si riesce ad avere un gradiente diverso da 0, il che poteva verificarsi se non si fosse preso il minimo tra i due termini.

2.2.2.2 *Pseudo-codice dell'algoritmo*

Come si può notare dallo pseudo codice dell'algoritmo qui sotto presentato, lo stile con cui vengono raccolte le informazione per addestrare la policy è quello degli Actor-Critic, ovvero un insieme di $\mathbb{N}$ attori che vengono eseguiti in contemporanea su environment diversi utilizzando la policy $\pi_{\theta_{old}}$. La novità importante sta però nell'istruzione alla penultima riga dell'algoritmo, vi è infatti riportato che l'ottimizzazione del

surrogato $L(\theta)$ viene fatta per un numero K di epoche, utilizzando minibatch di dimensione $M \leq NT$. Questo è particolarmente desiderabile in quanto permette di imparare di più da ogni informazione ottenuta, migliorando la sample-efficiency dell'algoritmo. Questo procedimento è pero adottabile solamente grazie agli accorgimenti presi, evidenziati nella sezione precedente, nel formalizzare la funzione obiettivo. Nei metodi policy gradient vanilla tutto ciò non sarebbe possibile, in quanto la policy cambierebbe troppo tra un passo di update e l'altro.
Se si riflette sul significato matematico dell'equazione 15 e sul processo

Algorithm 3: PPO, in stile Actor-Critic

```
for iteration=1,2,... do
    for actor=1,2,...,N do
        Seguire la policy π_{θ_old} nell'ambiente per T timesteps;
        Calcolare la stima degli Advantage Â_1, ..., Â_t;
    end
    Ottimizzare il surrogato L(θ) per K epoche e dimensione
     minibatch M ≤ NT θ_old ← θ
end
```

di ottimizzazione in più epoche per ogni iterazione risulterà evidente che alla prima epoca π sarà uguale a pi_{old} per cui nessun aggiornamento verrà tagliato, il che significa che è garantito che qualcosa verrà appreso dagli esempi raccolti. Dalla seconda epoca in poi la nuova policy potrebbe differire dalla vecchia e di conseguenza potrebbe anche raggiungere i limiti fissati, causando l'andamento a 0 del gradiente per quegli esempi che portano la policy troppo distante rallentando l'addestramento per questa iterazione fino a quando non si passa alla successiva e verranno raccolti nuove esempi su cui fare training.
Per completezza bisogna specificare che al fine di calcolare la stima della Advantage function si fa spesso uso di state-value function $V(s)$ appresa. Se, come nel lavoro di Schulman et al., si fa uso un'architettura di rete neuronale in cui la policy e la value function hanno parametri condivisi, bisogna utilizzare una funzione di loss che combini entrambe le cose.
Come spesso accade, per aumentare la capacità di esplorare lo spazio delle soluzioni viene introdotto un bonus per premiare l'entropia. La funzione risultante è quindi:

$$L_t^{CLIP+VF+S}(\theta) = \hat{E}_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF} + c_2 S[\pi_\theta](s_t)] \tag{16}$$

Dove c_1, c_2 sono dei coefficienti, S indica il bonus entropia e L_t^{VF} la loss quadratica $(V_\theta(s_t) - V_t^{targ})^2$.
Per la stima dell'advantage si usa l'equazione:

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + ... + ... + (\gamma\lambda)^{T-t+1}\delta_{T-1} \tag{17}$$

$$dove\ \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{18}$$

2.2.3 *Tipologie di PPO*

Nell'articolo di presentazione di PPO vengono in realtà presentata due possibili approcci:

1. **PPO-Clip:** si tratta dell'approccio descritto in tutta la sezione 2.2, non prevede nessun vincolo, bensì dei particolari accorgimenti nella funzione obiettivo per rimuovere l'incentivo a rendere la nuova policy troppo diversa dalla precedente.

2. **PPO-Penalty**: si tratta di un approccio alternativo presentato contestualmente all'altro, esso, similmente a TRPO, compie l'aggiornamento della policy ponendo un limite alla KL-Divergence. In questo caso però non si ha un vincolo da rispettare, bensì un coefficiente di penalità aggiustato durante il training.

Il secondo approccio, a parere degli autori, potrebbe venire utilizzato come alternativa, o anche congiuntamente, al primo. L'idea è di avere una penalità basata sulla KL-Divergence ed adattare un coefficiente di penalità in modo da raggiungere un certo target di KL-Divergence ad ogni aggiornamento
Nella più semplice istanziazione di questo approccio mostrata nell'articolo per ogni step di aggiornamento della policy si seguono i seguenti passi:

1. Si utilizza la discesa del gradiente stocastica per diverse epoche al fine di ottimizzare la funzione obbiettivo:

$$L^{KLPEN}(\theta) = \hat{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t - \beta KL[\pi_{old}(\cdot|s_t), \pi_\theta(\cdot|s_t)]\right] \tag{19}$$

2. Calcolare $d = \hat{E}_t[KL[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)]]$
   - Se $d < d_{targ}/1.5,\ \beta \leftarrow \beta/2$
   - Se $d > d_{targ} \times 1.5,\ \beta \leftarrow \beta \times 2$

Nell'articolo si precisa che i valori 1.5 e 2 sono stati scelti euristicamente, anche se in realtà l'algoritmo non è troppo sensibile ad essi.

2.2.3.1 *Comparazione dei due approcci*

Nell'articolo di Schuman et al. viene fatta un interessante comparazione fra i risultati ottenuti con i diversi approcci da loro proposti.
Viene quindi riportato che nella comparazione, anche per facilitare la ricerca dei migliori iper-parametri, sono stati utilizzare dei benchmark computazionalmente poco onerosi, in particolare sono stati scelte 7 task robotici simulati[1] implementati nell'ambiente OpenAI Gym [5], il quale

1 2HalfCheetah, Hopper, InvertedDoublePendulum, InvertedPendulum, Reacher, Swimmer, Walker2d

fa uso del motore fisico MuJoCo[46].
Per ogni task vengono eseguiti 1 milione di step di training, utilizzando una rete totalmente connessa MLP con due strati nascosti composti da 64 unità e funzione di attivazione tanh. In questo caso non vengono condivisi parametri della rete tra la policy e la value function e non viene utilizzato il bonus per l'entropia.

Algoritmo	Punteggio medio normalizzato
No clipping o penalità	-0.39
Clipping, $\epsilon = 0.1$	0.76
Clipping, $\epsilon = 0.2$	**0.82**
Clipping, $\epsilon = 0.3$	0.70
KL adattivo $d_{targ} = 0.003$	0.68
KL adattivo $d_{targ} = 0.01$	0.74
KL adattivo $d_{targ} = 0.03$	0.71
Kl fisso, $\beta = 0.3$	0.62
Kl fisso, $\beta = 1$	0.71
Kl fisso, $\beta = 3$	0.72
Kl fisso, $\beta = 10$	0.69

Tabella 1: Risultati del confronto tra varie versioni di PPO con varianti in base ai parametri ϵ, d_{targ}, e β. Risultati tratti da [38]

I valori riportati nella tabella 1 sono il risultato di una serie di operazioni per far in modo che il punteggio ottenuto da una policy random equivalga a 0 ed il punteggio massimo ottenuto equivalga ad 1. Il punteggio è quindi la media normalizzata dello score ottenuto negli ultimi 100 episodi.
Come già anticipato il risultato migliore è quello totalizzato da versione di PPO-Clip, utilizzando come valore di ϵ 0.2.

2.3 ESEMPI APPLICATIVI E CONFRONTO CON ALTRI ALGORITMI

Al fine di evidenziare la bontà dell'approccio, gli autori di PPO propongono alcuni confronti con algoritmi allo stato dell'arte, che verranno mostrati in questa sezione.

2.3.1 *Comparazione in domini continui*

La versione migliore di PPO, ovvero quella Clipped con $\epsilon = 0.2$ viene confrontata i seguenti algoritmi:

- Trust Region Policy Optimization[37](TRPO)
- Cross-entropy method (CEM)[44]

- Vanilla policy gradient con stepsize adattivo
- A2C (versione sincrona di A3C[25])
- A2C con trust region [49]

Il terreno di confronto è lo stesso utilizzato negli esperimenti per confrontare le varie versioni di PPO.
Come evidenziato dai grafici nella figura 8 PPO si dimostra migliore

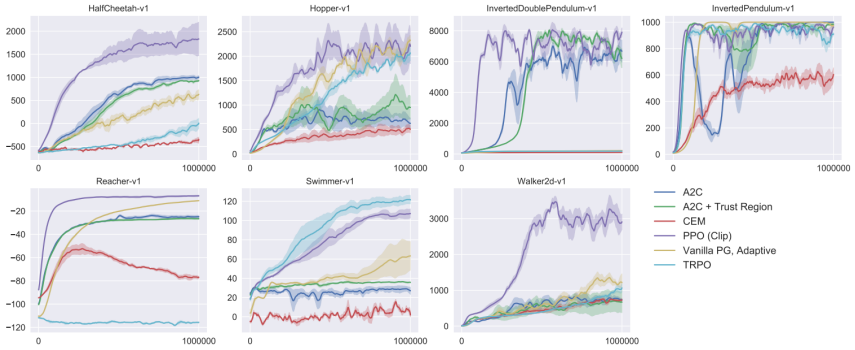


Figura 8: Grafici di confronto delle prestazione degli algoritmi presi in esame in 7 task presenti in OpenAI Gym. Tratta da [38]

in 5 task sui 7 testati e nei restanti due ottiene performance vicine al migliore.

2.3.2 *Comparazione sui giochi Atari*

Vista la centralità dei giochi Atari nelle pubblicazione nel campo del reinforcement learning negli ultimi anni, gli autori hanno voluto misurare le prestazioni di PPO anche su questo terreno.
In questo caso il confronto è stato effettuato contro gli algoritmi A2C ed ACER, considerati allo stato dell'arte. I tre algoritmi sono stati quindi testati su 49 giochi Atari, venendo addestrati per 40 milioni di frame di gioco, che si traducono in 10 milioni di timestep (solitamente vengono presi 4 frame per volta come observation).

	A2C	ACER	PPO	Pareggio
Reward medio per episodio su tutto addestramento	1	18	30	
Reward medio per episodio su ultimi 100 episodi	1	28	19	1

Tabella 2: Tabella del numero di giochi in cui ciascun algoritmo ha ottenuto il punteggio migliore, diviso per metrica di giudizio, il numero totale di giochi è 49. Risultati tratti da [38]

Nella tabella 2 vengono evidenziati i risultati di questa comparazione.
Per confrontarli vengono utilizzate due metriche di giudizio:

1. Reward medio per episodio calcolato su tutto l'addestramento. In questo caso vengono favoriti gli algoritmi che mostrano un addestramento più rapido.

2. Reward medio per episodio calcolato sugli ultimi 100 episodi. In questo caso vengono invece premiati gli algoritmi che hanno una miglior performance alla fine dell'addestramento.

Come si evince dalla tabella riportata PPO è senz'altro più veloce nell'addestramento rispetto agli altri due algoritmi, seppur si dimostra peggiore in termini di performance finale. ACER ottiene il risultato migliore nel 57% dei giochi, mentre PPO nel 38%, va però considerato che ACER è un algoritmo molto più pesante e complesso, sia a livello computazionale che di implementazione.

2.3.3 *PPO per controllare un umanoide 3D*

Un problema non banale su cui testare un algoritmo di RL è il pilotaggio degli attuatori di un robot umanoide. In questo caso viene utilizzato un simulatore, chiamato Roboschool (sempre nell'ambito di OpenAI Gym), ma il problema resta molto complesso, in quanto il robot umanoide in questione è composto da 17 giunture a cui sono applicabili valori continui di forze.
Vengono quindi presi in considerazione tre task:

1. **RoboschoolHumanoid**: l'umanoide deve solo imparare a camminare in avanti.

2. **RoboschoolHumanoidFlagrun**: l'umanoide deve saper dirigersi verso un obiettivo, il quale cambia posizione ogni 200 timestep oppure quando viene raggiunto.

3. **RoboschoolHumanoidFlagrunHarder**: l'obiettivo è lo stesso del punto precedente ma il robot viene occasionalmente colpito da dei cubi che lo fanno rovinare a terra.

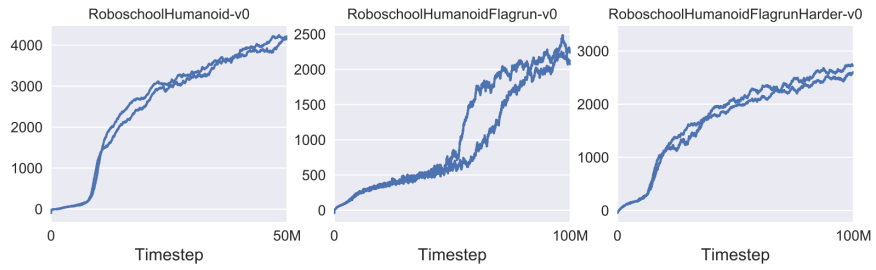


Figura 9: Grafici delle curve di apprendimento di PPO applicato ai tre task presentati. Tratta da [38]

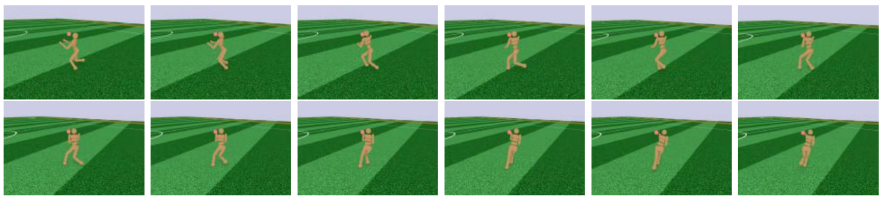


Figura 10: Sequenza di 14 frame del comportamento appreso dall'umanoide nel task RoboschoolHumanoidFlagrun. Tratta da [38]

L'algoritmo si dimostra capace di migliorare la propria performance nel tempo, con una curva di apprendimento piuttosto buona, come evidenziato in figura 9. Ovviamente il 2° ed il 3° task sono molto più difficili, a riprova di questo fatto si noterà dal grafico che richiedono il doppio dei timestep per raggiungere buoni risultati pur utilizzando solo 32 attori per il task 1 a fronte dei 128 usati negli altri 2

A dimostrazione della bontà del risultato si può notare ciò che accade nella sequenza di frame riportati nella figura 10: nei primi 6 frame il robot corre verso un certo target, la sua posizione viene poi cambiata e quindi esso si orienta verso il nuovo target.

3

LA COMPETIZIONE ANIMAL-AI OLYMPICS

In questo capitolo verrà descritta nel dettaglia la competizione Animal-AI Olympics[3] a cui si è deciso di prender parte per saggiare il reinforcement learning in un ambito applicativo molto sfidante.
Innanzitutto verranno evidenziate le motivazioni e le idee che stanno dietro l'ideazione di questa competizione da parte degli organizzatori per poi introdurre brevemente i framework tecnologici in cui è stato sviluppato l'ambiente della competizione.
Nella terza sezione verranno infine approfonditi tutti gli aspetti della competizione, analizzando con precisione ambiente, possibili oggetti, modalità di valutazione ed in generale fare quindi una seria analisi del dominio applicativo, in modo da poter poi sviluppare un progetto per attaccare il problema.

3.1 INTRODUZIONE ALLA COMPETIZIONE

Come già evidenziato in altre sezioni i giochi ricoprono un ruolo centrale nel dimostrare le capacità (o le debolezze) degli approcci di RL esistenti, il motivo per cui essi vengono tipicamente scelti come ambiti dimostrativi può essere certamente ricondotto ad alcune motivazioni principali:

- **Accessibilità e versatilità**: disporre di un ambiente virtuale, facilmente utilizzabile ed uguale per tutti è di grandissimo aiuto per poter fare ricerca e per confrontare le soluzioni
- **Transizioni di stato ben definite:** sopratutto nei giochi più semplici è facile identificare quando si passa da uno stato all'altro del gioco, in più molto spesso gli ambienti sono deterministici, facilitando anche la riproduzione degli esperimenti.
- **Obiettivi ben precisi ed identificabili:** in qualsiasi gioco è possibile trovare un obiettivo preciso o un punteggio da massimizzare. Ciò è fondamentale nell'ambito dell'apprendimento per rinforzo ed allo stesso tempo può fornire una metrica di giudizio oggettiva per confrontare i risultati.
- **Popolarità:** anche se meno importante delle prime tre motivazioni, è comunque un aspetto da non trascurare. Infatti, vedere un algoritmo che riesce a superare le performance umane su dei

giochi conosciuti garantisce sempre un certo impatto, creando maggior interesse nella ricerca.

Per questi, ed anche altri, motivi sono sorti una miriade di framework di test per algoritmi di reinforcement learning, sia relativi a giochi sia a simulatori fisici.
Nonostante gli impressionanti risultati raggiunti da vari approcci in specifici giochi/task, la maggior parte di essi hanno un difetto comune: sono capaci di fare bene il solo compito per cui sono stati addestrati[32], anche se vengono provati su due task che richiedono abilità simili. A questo proposito è emblematica questa frase tratta dall'articolo di presentazione della competizione [3]:

> *Humans are no longer the best Go players, but they are still the best at functioning across a wide number of environments and adapting to unexpected situations*

Ritengo che questa frase descriva straordinariamente lo stato corrente della disciplina del reinforcement learning ed in generale dell'AI, infatti non si può negare che in specifici ambiti questi algoritmi abbiano raggiunto performance ottime, ma al contempo non esiste nessun approccio che abbia una anche solo discreta capacità di generalizzare ed astrarre concetti.
Il dibattito su quanto la disciplina sia lontana da raggiungere le capacità umane, ovvero arrivare alla famosa Artificial General Intelligence (AGI), è in realtà molto fervente, c'è chi crede questo traguardo verrà tagliato in una decina di anni e chi invece il nastro d'arrivo lo pone tra una quarantina di anni. Collegata alla domanda quando (e se) l'umanità arriverà alla AGI vi è una questione più tecnica: con quale approccio ci si può arrivare?
Bisogna infatti evidenziare come i recenti successi nell'ambito del Deep Learning abbiano portato molti a pensare che è questa la via maestra per arrivare al traguardo tanto agognato. Al contrario altri pensano che non sia possibile inquadrare la complessità della mente umana nella cornice del Deep (Reinforcement) Learning, bensì bisognerà unire più paradigmi.
A prescindere da come la si pensi riguardo a questi complessi interrogativi è innegabile che nella stra-grande maggioranza dei casi gli algoritmi di RL vengano testati e valutati in condizioni molto simili a quelli di training. Ovviamente con questo non si vuole intendere che gli agenti vengano valutati negli stessi identici task visti durante il training, ma che questi scenari vengano estratti dalla stessa distribuzione, impedendo di verificare l'effettiva capacità di generalizzazione.
Concludendo, al fine di verificare meglio quali siano le capacità reali di un agente (sperabilmente) intelligente, servirebbe un ambiente di test che offra vantaggi paragonabili a quelli dei giochi che solitamente si utilizzano già, ma al contempo garantendo che solo un agente realmente capace di generalizzare possa ottenere performance umane.

3.1.1 *Perché Animal-AI?*

Leggendo il nome della competizione ci si potrebbe chiedere perché gli autori hanno deciso di accostare il mondo animale a quello dell'intelligenza artificiale.
La risposta sta nel fatto che l'umanità da sempre si è posta alcune domande fondamentali: come funziona la coscienza? Come si possono misurare le abilità cognitive? Come funziona la mente delle altre specie viventi? Alcuni animali dispongono di capacità cognitive simili alle nostre?
Ovviamente molte di queste domande sono tutt'altro che banali e rispondere esaustivamente ad esse richiederà ancora molto tempo, ma è innegabile che negli ultimi 100 anni molto è stato fatto, ad esempio nello studiare le capacità cognitive degli animali, sviluppando tutto un insieme di test per esaminare e misurarne le effettive capacità.
I test che sono stati ideati in tutti questi anni di ricerca sulla cognizione animale sono ormai molteplici e possono variare da task molto semplici quali verificare una comprensione di base del concetto di distanza oppure molto complessi come ad esempio quelli che includono una serie non banale di relazioni causa-effetto.
Molto spesso questi test, sopratutto quando si vogliono fare comparazioni, richiedono una progettazione attenta ed un metodo rigoroso, al fine di valutare al meglio le capacità prese in esame. Solitamente un esperimento di questo tipo dovrà quindi rispettare 3 principi fondamentali(da [3]):

- La risoluzione del task richiede delle capacità di base date per assodate, come la capacità di muoversi, riconoscere forme e colori, e desiderare del cibo.
- La capacità cognitiva che si sta cercando di dimostrare deve essere posseduta per superare il task.
- Tutti gli eventuali fattori che possono influenzare il test devono essere eliminati, non ci deve perciò essere incertezza sul fatto che se il task è stato superato è perché l'animale possiede quella determinata capacità ricercata.

Scopo principali di questi test è quindi non solo scoprire quali compiti riesce a svolgere un certo animale, ma sopratutto come egli riesce a farlo e come esso riesca a ragionare ed interpretare quello che succede nel mondo che lo circonda.
L'idea degli organizzatori della competizione è stata quindi di applicare le metodologie dell'analisi delle capacità cognitive degli animali nel testare degli algoritmi di intelligenza artificiale. Questo nasce da una domanda che può sorgere molto spontanea, ovvero, *può lo stato dell'arte dell'AI superare le prove che gli animali hanno dimostrato di eccellere*? Ed ancora più importante, *come affronteranno queste sfide gli algoritmi*

messi in campo?
Fortunatamente molti dei test concepiti per gli animali veri e propri possono essere tradotti in una versione informatica, applicabile ad algoritmi di intelligenza artificiale. Da qui nasce l'idea di organizzare una competizione mondiale al fine di dimostrare la capacità di un agente artificiale di superare dei task che richiedono una serie di capacità cognitive.

3.1.2 *Organizzatori della competizione*

La competizione Animal-AI Olympics è stata organizzata da un comitato composto da:

- **Dr. Matthew Crosby** - Imperial College London - Leverhulme CFI Postoctoral Researcher
- **Benjamin Beyret** - Imperial College London - Leverhulme CFI Research Assistant
- **Prof. Murray Shanahan** - DeepMind - Imperial College London, Leverhulme CFI Spoke Leader
- **Dr. Marta Halina** - University of Cambridge - Leverhulme CFI Project Leader
- **Dr. Lucy Cheke** - University of Cambridge - Leverhulme CFI Associate Fellow
- **Prof. José Hernández-Orallo** - Universitat Politècnica de València - Leverhulme CFI Associate Fellow

La competizione è stata promossa, ideata e sponsorizzata dal Leverhulme Centre for the Future of Intelligence di Cambridge, insieme alla società GoodAI ed all'Imperial College di Londra. Tra gli sponsor della competizione anche Amazon AWS, la Whole Brain Architecture Initiative e l'Artificial Intelligence Journal.

3.1.3 *Scopo della competizione*

La competizione Animal-AI Olympics si propone quindi di testare le abilità di un agente artificiale in diverse categorie di task, ognuna delle quali richiede un certo grado di capacità cognitiva per assolverla con successo.
L'ambito in cui la competizione è inquadrata è quello del Reinforcement Learning, ma ciò non significa che l'insieme degli strumenti di questo ambito sia necessariamente sufficiente per raggiungere performance ottimali.
Dovrebbe essere abbastanza evidente il fatto che tradurre dei test concepiti per animali in test per sistemi di intelligenza artificiale non

è banale, vanno perciò trovate delle modalità grazie alle quali si può astrarre dalle mere caratteristiche fisiche dell'animale, per concentrarsi sui tratti comune che accomunano la maggior parte delle specie. Perciò in generale tutti i test si baseranno su tre tratti principali:

- **Visione:** la capacità di vedere ciò che circonda l'agente
- **Navigazione:** l'abilità di muoversi in un ambiente, anche complesso.
- **Raccolta di cibo:** l'agente deve saper trovare del cibo (venendo così internamente premiato)

Aspetto particolare della competizione è che i test sono nascosti, per cui non si ha a disposizione le particolari configurazioni dell'ambiente in cui verrà testato l'agente al fine di determinare quali capacità possiede. Questo accorgimento è al fine di evitare che l'agente impari a superare un task senza esibire in realtà le capacità cognitive che sarebbero richieste.
In definitiva quindi gli organizzatori si propongono di mettere a disposizione un ambiente apposito dove saranno poi effettuati questi test, esso avrà le seguenti caratteristiche:

- L'agente agirà in un'arena di dimensioni fisse, con al suo interno un insieme di semplici oggetti, al fine di evitare di introdurre fattori di confusione nei test.
- La fisica modellata dovrà essere realistica, rispettando principi quali gravità, attriti e collisioni.
- Dovrà essere semplice implementare e riprodurre test cognitivi ed al contempo bisognerà garantire compatibilità con gli strumenti standard del machine learning.

3.2 L'AMBIENTE DI SIMULAZIONE

Prima di entrare nel dettaglio di tutti gli aspetti della competizione è bene parlare brevemente degli aspetti tecnologici dell'ambiente rilasciato dagli organizzatori, anche introducendo i due framework utilizzati.
L'ambiente viene fornito come un eseguibile interfacciabile tramite API Python, esso è stato sviluppato sulla base di **Unity ML-Agents**, il quale fornisce una serie di funzionalità molto comode per utilizzare un ambiente sviluppato in Unity nell'ambito del machine learning.
Sopra a questo livello è stato inoltre fornito un wrapper che implementa l'interfaccia di OpenAI Gym, permettendo una più facile integrazione con le baseline già esistenti e consolidate.
L'ambiente in questione è composto da una o più arene, in ognuna delle quali è presente un solo agente, totalmente indipendente da quelli presenti nelle altre arene.Riguardo a tutti gli aspetti riguardanti le possibili

configurazioni dell'arena di gioco e tutti gli aspetti della competizione essi verranno però presentati nella sezione successiva.

3.2.1 *Unity ML-Agents*

Unity Machine Learning Agents Toolkit [16] è un plugin Unity open-source il cui scopo è quello di permettere di sfruttare giochi e simulatori sviluppati in Unity per addestrare e testare agenti intelligenti.
Infatti grazie ad una API Python gli agenti possono essere controllati con l'approccio che più aggrada lo sviluppatore.
Molto interessante è anche la disponibilità di implementazioni di algoritmi allo stato dell'arte per il Reinforcement Learning, fornendo così uno strumento molto efficace sia per realizzare giochi con giocatori non controllabili sia per confrontare vari approcci.

3.2.1.1 *Componenti principali*

Il toolkit in questione contiene, ad alto livello, tre macro-componenti principali, come evidenziato in figura 11:

- **Learning Environment:** contiene la Unity Scene e gli agenti del gioco. A sua volta contiene due componenti che aiutano ad organizzare la Unity scene:
  - **Academy:** la quale si occupa di gestire il ciclo di raccolta delle observation e di decisione delle azioni da intraprendere. Al suo interno vengono gestiti molti parametri relativi all'arena di gioco, quali la qualità e la velocità di rendering.
  - **Agents:** ognuno dei quali è collegato ad uno specifico Unity GameObject e si occupa di generare le observation, eseguire le azione che gli vengono comandate ed assegnare un reward positivo o negativo quando appropriato. Come si evince dalla figura un agente è associato ad un **Brain** dal quale provengono le indicazione sulle azioni da intraprendere.
- **Python API:** questo componente contiene tutti gli algoritmi di machine learning che vengono utilizzati per l'addestramento. Esso non fa parte di Unity, infatti comunica esternamente tramite un apposito componente.
- **External Communicator**: è prorio il componente deputato alla comunicazione tra l'ambiente Unity e l'API Python.

3.2.1.2 *Possibili modalità di addestramento*

Data la flessibilità di questo toolkit, esistono varie modalità tramite il cui utilizzarlo per addestrare degli agenti di RL:

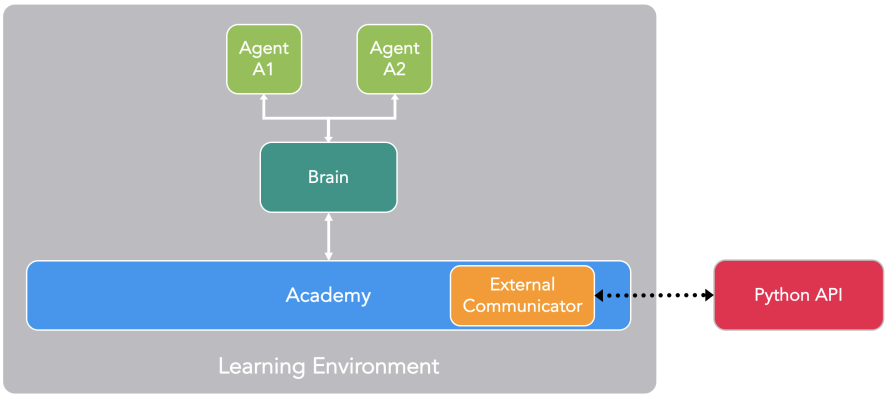


Figura 11: Schema di riferimento per gli ambienti sviluppati con Unity ML-Agents. Tratta da [16]

- **Addestramento ed inferenza con algoritmi built-in:** in precedenza si era già accennato che diversi algoritmi allo stato dell'arte vengono forniti nel toolkit. Questo risulta particolarmente comodo perché mentre la fase di addestramento deve essere necessariamente svolta includendo l'API Python, l'utilizzo dell'agente addestrato è invece già integrabile direttamente nell'agente Unity.

- **Addestramento di modelli custom**: se invece si vogliono usare algoritmi custom o comunque non quelli forniti direttamente da ML-Agents l'agente verrà controllato da codice Python sia in fase di addestramento che di inferenza. Molto interessante è anche la possibilità di utilizzare Curriculum Learning ed Imitation learning.

In generale il toolkit in questione, seppur abbastanza giovane, è già molto interessante e funzionale e può rappresentare sia un alternativa al più famoso OpenAI Gym ma anche essere complementare ad esso, in quanto come vedremo, un Unity ML Environment può implementare l'interfaccia di un Gym Environment.
La possibilità di usare uno strumento come Unity apre le porte alla realizzazione di scenari più complessi in cui testare i propri agenti di ML, si veda la figura 12 per dei possibili esempi di utilizzo a questo scopo.

3.2.2 *OpenAI Gym*

OpenAI Gym[5] è tra i più famosi toolkit per sviluppare e comparare algoritmi di reinforcement learning. La sua forza sta nel non fare nessuna assunzione sulla struttura degli algoritmi utilizzati ed è compatibile con qualsiasi libreria per il Machine Learning sviluppata su Python.
Tutti gli ambienti di test messi a disposizione hanno un interfaccia comune, permettendo quindi agli utilizzatori di testare i propri algoritmi in più scenari senza dover fare modifiche rilevanti al codice.

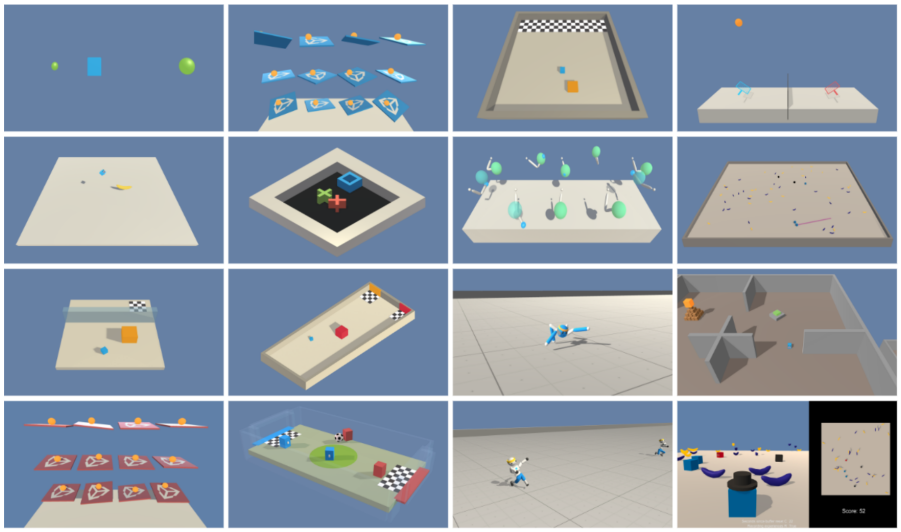


Figura 12: Alcuni esempi di ambienti sviluppati in Unity ed utilizzati per problemi di reinforcement learning. Tratta da [16]

3.2.2.1 *Motivazioni alla base*

Per gli ideatori di OpenAI Gym la ricerca nel campo del reinforcement learning, pur avendo raggiunto traguardi importanti, è rallentata da due principali fattori:

- **La necessità di migliori benchmark:** prima dell'introduzione di OpenAI Gym le soluzioni di benchmark open-source erano abbastanza scarne e molto spesso difficili da utilizzare.
- **Manca di standard negli ambienti utilizzati nella pubblicazioni:** nell'ambito del RL, anche se è un discorso che vale per molti altri casi, piccoli cambiamenti nel modo di definire dei concetti possono provare grandi discostamenti nei risultati finali. In generale quindi viene minata la riproducibilità e l'affidabilità dei risultati ottenuti.

Il toolkit in questione si propone quindi di superare queste limitazioni, fornendo interfacce comuni ed un ampia gamma di benchmark.

3.2.2.2 *Aspetti principali*

La forza di OpenAI Gym sta nella sua semplicità di utilizzo, esso infatti propone un interfaccia molto ristretta, permettendo una facile integrazione con qualsiasi tipo di algoritmo. Di seguito una tipica serie di episodi in un ambiente minimale come il "Cart-Pole".

```
import gym
env = gym.make('CartPole-v0')
for i_episode in range(20):
    observation = env.reset()
    for t in range(100):
        env.render()
```

```
        print(observation)
        action = env.action_space.sample()
        observation, reward, done, info = env.step(action)
        if done:
            print("Episode finished after {} timesteps".
    format(t+1))
            break
env.close()
```

Come si evince dal codice appena riportato per un ambiente gym esistono due metodi principali (più altri aggiuntivi):

- **reset():** il quale resetta lo stato dell'ambiente, restituendo l'observation iniziale di esso.
- **step(action):** questo metodo riceve in input l'azione da eseguire e restituisce in output 4 elementi:
  - **observation**: cioè un oggetto specifico dell'ambiente rappresentante quanto percepito dall'agente dopo aver eseguito l'azione in input.
  - **reward**: ovvero un punteggio ricevuto dall'ambiente dopo aver eseguito l'azione in input.
  - **done:** ovvero un flag che indica se l'ambiente va resettato di nuovo, nella maggior parte dei task infatti vi sono delle condizioni per cui l'episodio termina e per cui questo elemento assumerà valore true.
  - **info:** un dizionario contenente informazioni utili per il debugging.

Quindi da quanto presentato si evidenzia il rispetto del classico ciclo del reinforcement learning, come presentato nella sezione 1.2.1.
Altri concetti fondamentali sono quelli di ***action_space*** ed ***observation_space***, degli attributi definiti obbligatoriamente per ogni ambiente Gym e che descrivono il formato delle azioni e delle observation valide.

3.3 CARATTERISTICHE DELLA COMPETIZIONE

In questa sezione verranno analizzati estensivamente tutti gli aspetti dalla competizione, partendo innanzitutto dall'ambiente in cui gli agenti verranno testati, descrivendo i possibili oggetti ritrovabili in esso e valutando le caratteristiche cinematiche e ricettive dell'agente stesso. Una volta descritti con precisione questi aspetti relativi all'ambiente di simulazione si passerà ad aspetti più legati alla competizione in se, descrivendo la strutturazione della competizione in test in vari categorie, ognuna con le sue caratteristiche e difficoltà. Verranno poi prese in esame le modalità di valutazione e le fasi in cui è composta la competizione.

3.3.1 *Caratteristiche dell'arena e possibili oggetti*

Prima di passare ad analizzare tutte le caratteristiche dell'arena e degli oggetti in essa contenuti è bene specificare che essa è totalmente configurabile via file YAML, ad esclusioni di alcuni elementi che non possono variare, quali la dimensione dell'arena stessa e la presenza delle mura perimetrali. Di seguito un esempio di file di configurazione, il suo significato sarà più chiaro durante l'introduzione dei vari elementi.

```
!ArenaConfig
arenas:
  0: !Arena
    t: 0
    items:
    - !Item
      name: Wall
      positions:
      - !Vector3 {x: 10, y: 0, z: 10}
      - !Vector3 {x: -1, y: 0, z: 30}
      colors:
      - !RGB {r: 204, g: 0, b: 204 }
      rotations: [45]
      sizes:
      - !Vector3 {x: -1, y: 5, z: -1}
    - !Item
      name: CylinderTunnel
      colors:
      - !RGB {r: 204, g: 0, b: 204 }
      - !RGB {r: 204, g: 0, b: 204 }
      - !RGB {r: 204, g: 0, b: 204 }
    - !Item
      name: GoodGoal
```

3.3.1.1 *Dimensioni e coordinate*

L'arena, come mostrato in figura 13, è un quadrato di dimensioni 40x40, con delle mura perimetrali.L'origine è posta in basso a sinistra, ovvero al punto $(0, 0)$, è importante ricordare che in Unity quello che solitamente è l'asse y è invece l'asse z, per fare un esempio, un oggetti al centro dell'arena appoggiato per terra ha coordinate $(20, 0, 20)$ in Unity.
Il pavimento ha una sua texture specifica, così come le mura perimetrali. Infine l'agente può essere posizionato in qualsiasi punto (x, y, z) dell'arena con un qualsiasi orientamento α non occupato da un altro elemento, se non si specifica una posizione esso apparirà in una posizione casuale con un orientamento casuale.

3.3.1.2 *Possibilità di blackout e durata degli episodi*

Nella configurazione di un arena è possibile specificare innanzitutto quanto potrà essere lungo al massimo l'episodio. Ponendo questo parametro t a 0 si avrà un episodio di lunghezza infinita, a meno che non si incorra in oggetto che faccia terminare l'episodio. Se l'episodio ha

una durata limitata, ovvero con $T > 0$, l'agente riceverà ad ogni istante temporale una piccola penalità di valore $-1/T$. Nel caso in cui $T = 0$ questa penalità non viene inflitta.
Aspetto interessante è quello dei **blackout**, infatti tramite file di configurazione è possibile specificare quando (e se) avere degli istanti temporali nell'episodio in cui le luci sono spente, per cui l'agente non potrà fare affidamento sulla propria vista.
Nel caso si vogliano inserire questi blackout durante l'episodio è possibile farlo in due diverse modalità:

- **Blackout ricorrenti:** ogni t time-step vengono acceso o spente le luci dell'ambiente.
- **Blackout in specifici timestep:** con una lista di valori si indica quando tenere le luci spente/accese.

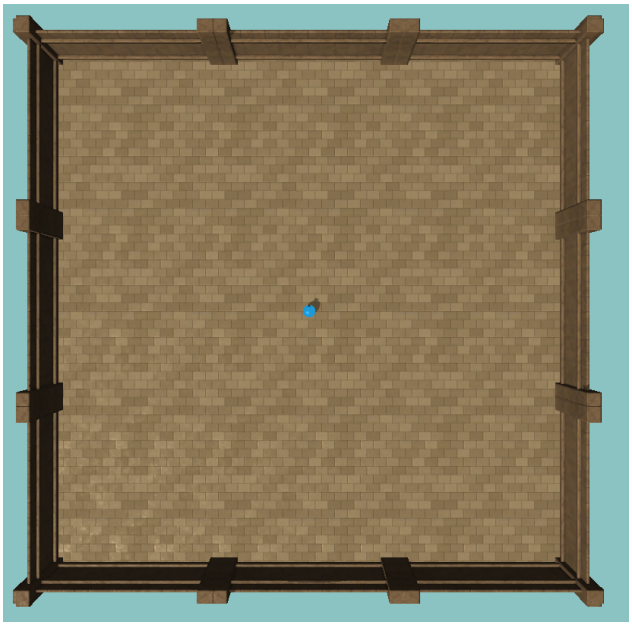
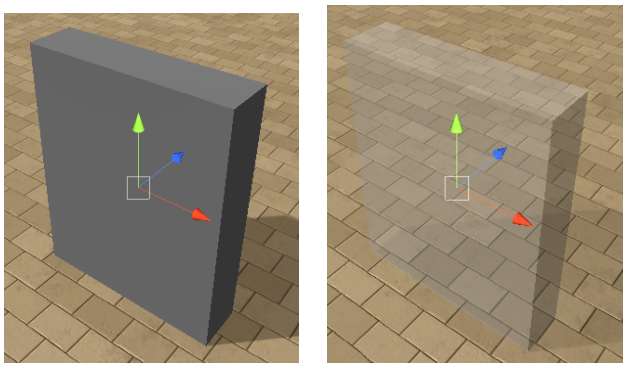


Figura 13: Arena di Animal-AI Olympics vuota con l'agente al suo centro. Tratta da [3]

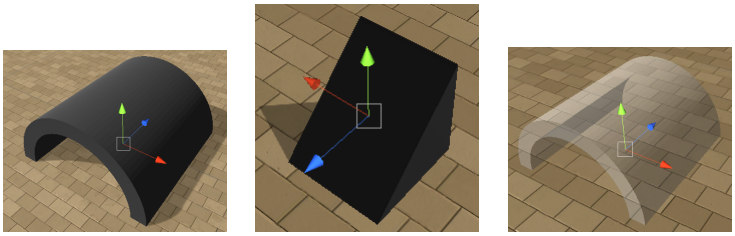
3.3.1.3 *Possibili oggetti*

I possibili oggetti che si possono trovare negli ambienti della competizione possono essere divisi in tre categorie principali:

- **Oggetti fissi:** di questa categoria fanno parte tutti quegli oggetti che non possono essere spostati dall'agente. Essi possono occludere la vista dell'agente o impedirgli di passare in certe porzioni dell'arena, dando quindi la possibilità di disporli in modo che l'agenti trovi una strategia per esplorare l'ambiente al fine di trovare del cibo. Anche se sono in numero limitato, questo tipo di oggetti



(a) *Muro opaco* (b) *Muro trasparente*



(c) *Tunnel opaco* (d) *Rampa* (e) *Tunnel trasparente*

Figura 14: Tutti gli oggetti fissi ritrovabili in un arena

possono rappresentare un ottima base per rappresentare ambiente anche complessi. Questa tipo di oggetti sono rappresentati in figura 14 ed includono:

- **Muro opaco:** questo oggetto in realtà si presta a più funzioni, infatti nel caso esso venga utilizzato come piattaforma assume tipicamente il colore blu, mentre come muro il colore grigio. Si consideri però che in alcune categorie di test queste convenzioni cromatiche potrebbero non essere rispettate (se ne parlerà nell'apposita sezione dedicata ai test). Le sue dimensioni variano da $(0.1, 0.1, 0.1)$ a $(40, 10, 40)$, va poi considerato nel caso in cui il muro sia particolarmente sottile, ovvero con $(x \vee z) \sim 0.1$ è possibile che urtando con particolare velocità il muro esso possa cadere. Questi oggetti potranno essere utilizzati in vari modi, sia singolarmente che combinandoli insieme per formare strutture più complesse, come un labirinto.
- **Muro trasparente:** esso gode delle stesse proprietà del muro opaco, escludendo ovviamente la parte cromatica. Questo tipo di oggetto è particolare perché pur trattandosi di un muro, permette di vedere attraverso di esso, il che lo rende simile ad una sorta di vetro, in questo caso però di colore fissato.
- **Rampa:** essa ha dimensioni che possono variare nel range

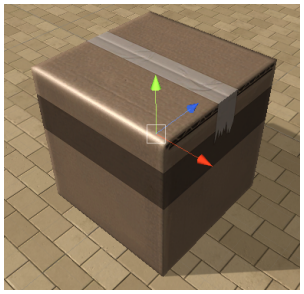
$(0.5, 0.1, 0.5)$ a $(40, 10, 40)$, nella maggior parte dei test avrà colore rosa. Questo oggetto ha l'ovvia funzionalità di far variare, in valore assoluto, la posizione sull'asse y.

  - **Tunnel:** ha dimensioni che possono variare nel range $(2.5, 2.5, 2.5)$ a $(10, 10, 10)$, solitamente vengono posti di colore grigio, salvo le solite eccezioni.
  - **Tunnel trasparente:** gode delle stesse proprietà della sua controparte opaca.

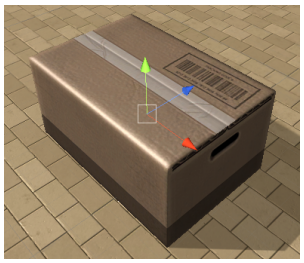
- **Oggetti spostabili:** come evidenziato dagli organizzatori[3], in psicologia comparativa agli animali esaminati è spesso richiesto di interagire con degli oggetti, magari spostandoli, afferrandoli, tirandoli o usandoli come strumenti. In un ambiente semplificato come quello proposto l'agente non può afferrare oggetti, ma grazie alla forze esercitata dalla sua velocità e massa può collidere con essi spostandoli. Va comunque considerato che essendo un simulatore dotato di una fisica realistica, anche altri oggetti in movimento possono spostare questi elementi dal basso peso. Esistono cinque incarnazioni di questa tipologia di oggetti, evidenziate in figura 15:
  - **Scatola di cartone leggera:** essa ha dimensioni che possono variare nel range $(0.5, 0.1, 0.5)$ a $(10, 10, 10)$ ed essendo molto leggera può essere spostata con relativa facilità da altri oggetti in movimento. Su di essa presenta una texture tipica degli scatoli da imballaggio.
  - **Scatola di cartone pesante:** essa ha dimensioni che possono variare nel range $(0.5, 0.1, 0.5)$ a $(10, 10, 10)$, ma risulta più pesante dell'altra e quindi più difficile da spostare. Si noti come la texture su di essa sia leggermente diversa da quella della scatola più leggere, così da permettere una più facile identificazione da parte dell'agente.
  - **Oggetto di legno a forma di U:** esso ha dimensioni che possono variare nel range $(1, 0.3, 3)$ a $(5, 2, 20)$, ha una texture in legno e la sua forma ad U, ovvero con due angoli retti all'estremità del segmento centrale. Questi angoli potrebbero essere sfruttati come appiglio dall'agente per poterli trasportare con maggiore facilità.
  - **Oggetto di legno a forma di L:**esso ha dimensioni che possono variare nel range $(1, 0.3, 3)$ a $(5, 2, 20)$, ha una texture in legno e la sua forma ad L, l'angolo retto può essere sfruttato come al punto precedente.
  - **Altro oggetto di legno a forma di L:** in questo caso la sua forma è simmetrica rispetto al punto precedente, ma valgono le stesse identiche considerazioni.

- **Ricompense:** questa tipologia di oggetti sono fondamentali, in quanto nel reinforcement learning sono l'unico feedback che l'agente riceve sulla qualità della propria performance.
  Dato lo spirito della competizione il parallelo tra reward positivi e cibo è abbastanza ovvio, così come i reward negativi possono essere visti come stimoli di repulsione, dolore o paura. Tutti gli oggetti/elementi che causano l'attribuzione all'agente di un punteggio (positivo o negativo) sono rappresentati in figura 16 e sono:
  - **Goal positivo stazionario:** si tratta di una sfera di colore verde con un raggio che va da 0.5 a 5.0, attribuisce all'agente un premio pari alla sua grandezza e fa terminare l'episodio.
  - **Goal positivo in movimento:** è in tutto simile alla sfera descritta al punto precedente ma si muove nell'arena, partendo da fermo puntando verso una certa direzione specificata nel file di configurazione o scelta a caso.
  - **Goal negativo stazionario:** si tratta di una sfera di colore verde con un raggio che va da 0.5 a 5.0, attribuisce all'agente una penalità pari alla sua grandezza e fa terminare l'episodio.
  - **Goal negativo in movimento:** è in tutto simile alla sfera descritta al punto precedente ma si muove nell'arena, valgono quindi le considerazioni fatte per il goal positivo in movimento.
  - **Reward positivo stazionario:** si tratta di una sfera di colore oro con un raggio che va da 0.5 a 5.0, attribuisce all'agente un premio pari alla sua grandezza ma non fa terminare l'episodio, se però nell'arena sono presenti solo sfere di questo tipo l'episodio termina quando tutte vengono raccolte (oppure ovviamente quando scade il tempo, se fissato). Questa tipologia di oggetto è molto importante perché permette di raccogliere più sfere e quindi, in molti casi, più punteggio del solito, perché l'episodio non viene subito terminato come accade con le sfere verdi.
  - **Reward positivo in movimento:** valgono le stesse considerazioni del punto precedente e delle sfere in movimento in generale.
  - **Zona mortale:** è rappresenta da un riquadro rosso di dimensioni che possono variare da $(1, 0, 1)$ a $(40, 0, 40)$, passare sopra di esse provoca la fine immediata dell'episodio e l'attribuzione di una penalità pari a -1. Queste zone sono sempre piatte e locate sul terreno.
  - **Zona calda:** è rappresenta da un riquadro arancione di dimensioni che possono variare da $(1, 0, 1)$ a $(40, 0, 40)$, passare sopra di esse provoca l'attribuzione di una penalità

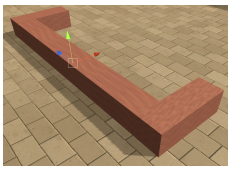
pari a $min(-10/T, -0.00001)$ o -0.00001 se $T = 0$, con T numero di step massimi dell'episodio, ma non fa terminare l'episodio. Queste zone sono sempre piatte e locate sul terreno.



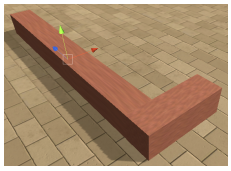
(a) *Scatola di cartone leggera*



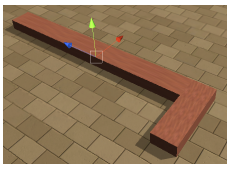
(b) *Scatola di cartone pesante*



(c) *Oggetto di legno a forma di U*



(d) *Oggetto di legno a forma di L*



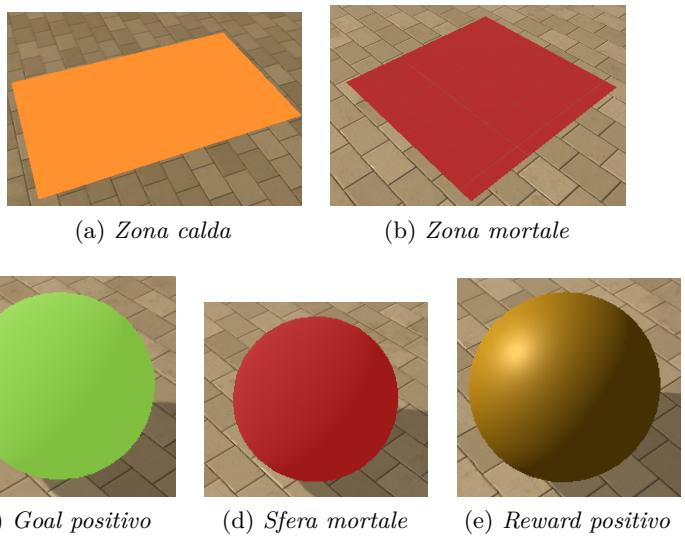
(e) *Altro oggetto di legno a forma di L*

Figura 15: Tutti gli oggetti spostabili ritrovabili in un arena

3.3.1.4 *Riassunto delle caratteristiche dell'arena*

Nell'arena possono essere presenti ostacoli fissi e/o movibili reward positivi e/o negativi. Solitamente i reward positivi e negativi sono rappresentati da sfere di un determinato colore, ma esistono anche delle zone dell'arena che forniscono un reward negativo o addirittura fanno terminare l'episodio con una penalità. Un episodio si conclude se si verifica una di queste condizione:

- Si è superato il numero massimo di step
- Sono stati raccolti tutti i reward positivi
- Si è entrati in una death zone o si è colpita una sfera di colore rosso.



(a) *Zona calda* (b) *Zona mortale*

(c) *Goal positivo* (d) *Sfera mortale* (e) *Reward positivo*

Figura 16: Tutte le zone e tutte le sfere che attribuiscono un punteggio positivo/negativo all'agente.

3.3.2 *Modello di movimento dell'agente*

L'agente, come visibile in figura 13, è rappresentato come una sfera di color blu, con un raggio di 0.5, su di esso è visibile un pallino nero che ne indica l'orientamento frontale.
Le azioni che esso può eseguire sono molto semplici e riguardano solamente il suo spostamento, esse sono organizzate su due dimensioni con 3 possibili opzioni ciascuna:

- La prima componente indica se andare verso avanti (1), se verso dietro (2) o niente(0).
- La seconda componente indica se ruotare verso destra (1), se verso sinistra (2) o non ruotare (0).

E' bene evidenziare che queste due componenti non sono una esclusiva dell'altra, bensì si compongono, le si può infatti pensare come due forze applicate all'agente. Se, ad esempio, si sceglie di far eseguire l'azione [1,2] all'agente ciò significa che esso aumenterà la sua velocità sull'asse z e contemporaneamente ruoterà verso sinistra.
Per quanto riguarda la rotazione che compie l'agente (nel caso la seconda componente assuma valore 1 o 2), essa è quantificabile in 6° nella direzione indicata. Da ciò ne deriva, banalmente, che per compiere un intero giro su se stesso l'agente abbia bisogno di 60 step, questa considerazione è importante sopratutto alla luce del fatto che molto spesso gli episodi hanno una durata finita, con un numero di step relativamente basso (250,500 o 750).
L'agente può avere una velocità massima di 21.13 m/s, velocità che può raggiungere accelerando in avanti per circa 53 timestep. Un grafico

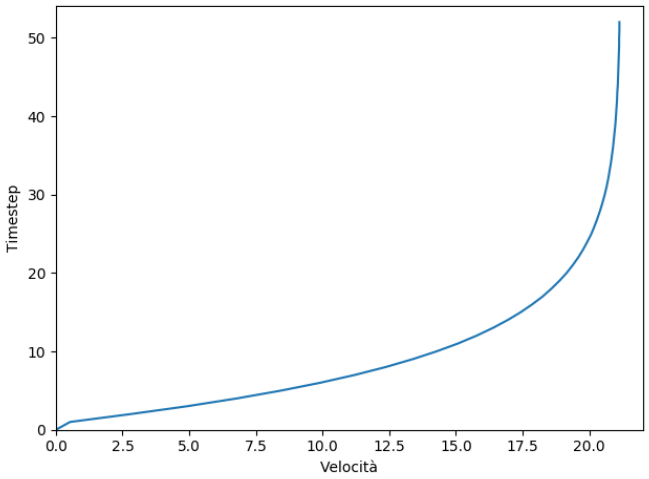


Figura 17: Grafico della velocità sull'asse Z di un agente che parte da fermo e si muove in avanti per 54 step

della velocità dell'agente partendo da fermo e andando sempre dritto per la massima distanza possibile nell'area è mostrato in figura 17.

3.3.3 *Capacità percettive*

L'agente in questione ha tre specifiche modalità di percepire il mondo che lo circonda e gli effetti delle proprie azioni: attraverso **la vista**, attraverso la percezione di **velocità** ed attraverso la ricezione di **premi/penalità**.
Di come vengono attribuiti i reward positivi e negativi si è già parlato nelle sezioni precedenti, per cui verranno ore considerate le altre due capacità percettive.

3.3.3.1 *La vista*

L'agente è dotato di vista monoculare, per cui ad ogni timestep acquisisce perciò un'immagine di ciò che ha di fronte. L'angolo di visione di questo sensore di acquisizione è di 60°. L'immagine è composta semplicemente da una matrice di pixel RGB dalle dimensioni di 84x84.
In realtà in fase di addestramento si può utilizzare una risoluzione maggiore, fino a 512x512 dell'immagine acquisita, però al momento del test l'unica risoluzione consentita è 84x84.
Si ricorda che in alcuni momenti l'agente potrebbe ricevere un immagine completamente nera a causa dei Blackout nell'arena.

3.3.3.2 *La percezione delle velocità*

L'agente, diversamente da quanto avviene negli umani, i quali percepiscono l'accelerazione, percepisce la sua velocità. E' importante specificare che le velocità percepite hanno un sistema di riferimento

basato sull'agente e non quello assoluto dell'arena.
Se immaginiamo l'agente nella posizione (0,0) dell'arena, ovvero nell'angolo in basso a sinistra, orientato con la fronte in avanti, ovvero sull'asse z dell'arena, il suo sistema di riferimento è coerente con quello dell'arena, con l'asse x che punta verso la sua destra e l'asse y verso l'alto. Nel caso generale però questo sistema di riferimento è basato sulla direzione dell'agente, per cui l'asse z punta sempre nella direzione frontale dell'agente, con gli altri assi di conseguenza.
Questa osservazione è molto importante sopratutto se si desidera mappare i movimenti dell'agente, anche considerato che di fatto l'agente non conosce la propria posizione e orientamento iniziale.
Dato che si dispone della velocità dell'agente sui tre assi si può calcolare lo spazio percorso, utilizzando la classica formula $s = v * t$, in questo caso t assume il valore empirico di 0.059.

3.3.4 *Categorie di test*

A prescindere dalla modalità di valutazione e dalla varie fasi della competizioni, le quali verranno trattati nella prossima sezione, esiste una netta ed importante distinzione in categorie dei task in cui gli agenti verranno valutati. La divisione in categorie è importante per indicare quanto un agente possieda la capacità cognitiva necessaria per superare i task di quella categoria, dando origine ad una sorta di profilo cognitivo.
Le dieci categorie non hanno tutte lo stesso grado di difficoltà, si può infatti tracciare una linea di separazione fra categorie più semplici e meno innovative, ovvero le prime quattro, e le più complesse e sfidanti, cioè le ultime sei.
Sebbene le categorie più semplici non appaiano particolarmente interessanti, in realtà per ottenere un ottimo punteggio su tutte e quattro l'agente dovrà esibire delle capacità di generalizzazione non banali.
Sebbene i test siano nascosti, gli organizzatori hanno voluto fornire delle descrizioni riguardo alle caratteristiche della varie categorie in cui gli agente verranno testate, esse sono quindi riportate di seguito:

1. **Procacciamento di cibo**: solitamente l'ottenimento di cibo è un aspetto presente nei test cognitivi che riguardano gli animali, essi hanno infatti una naturale propensione nell'andare verso le fonti nutritive. In questa categoria viene testata la capacità dell'agente di raccogliere le sfere verdi e gialle in arena con solo le varie sfere come possibili oggetti presenti. L'agente dovrà quindi sapere che le sfere gialle e verdi sono da raccogliere mentre quelle rosse vanno evitate. In questa categoria le principali difficoltà potrebbero essere legate a sfere di dimensioni molto ridotta che potrebbero essere non viste con facilità dall'agente. Altro elemento sensibile potrebbe essere il movimento di alcune sfere, che richiederà un apposita reazione da parte dell'agente.

2. **Preferenze:** questa categoria mira a valutare l'abilità dell'agente di scegliere la sequenza di azioni migliori più premiante. Ovviamente questo può variare dal preferire più cibo rispetto a meno cibo, oppure cibo ottenibile più facilmente piuttosto che difficilmente. L'agente dovrebbe quindi capire che sfere di diverse dimensioni attribuiscono anche punteggi diversi e che è preferibile ottenere molto più cibo tra poco piuttosto che poco cibo immediatamente. In questo caso nell'arena possono essere presenti tutte le tipologie di oggetti meno le zone calde e quelle mortali.

3. **Ostacoli:** in questa categoria l'arena può contenere barriere fisse, piattaforme e vari tipi oggetti spostabili e non che potrebbero ostacolare la navigazione dell'ambiente alla ricerca di cibo da parte dell'agente. La capacità di esplorazione è infatti identificata come un'abilità chiave di molti animali. Oltre a navigare intorno ai vari oggetti fissi per cercare il cibo, potrebbe essere anche necessario spostare qualche oggetto per completare tutti i task. In alcuni casi potrebbe essere indispensabile l'uso di rampe per superare ostacoli insormontabili. Anche in questo caso nell'arena possono essere presenti tutte le tipologie di oggetti meno le zone calde e quelle mortali.

4. **Evitare penalità:** in questo caso viene misura la capacità dell'agente di individuare ed evitare le zone che gli infliggono delle penalità. In questo caso l'agente dovrà eseguire una serie di task in cui dovrà però necessariamente minimizzare il tempo trascorso sulle zone arancioni ed evitare completamente quelle rosse. In questo caso tutti gli oggetti possono essere presenti nell'arena.

5. **Ragionamento Spaziale:** Questa categoria ha l'obbiettivo di testare l'abilità dell'agente di comprendere la spatial affordance dell'ambiente, ovvero come sfruttare le caratteristiche fisiche dell'arena per raggiungere il proprio obbiettivo, che ovviamente rimane quello del procacciamento di cibo. Per superare i task di questa categoria sarà necessario che l'agente capisca la fisica con cui funziona l'ambiente, ma anche ricordare le porzioni dell'arena già visitate.

6. **Generalizzazione:** Include variazioni dell'ambiente che possono sembrare superficialmente diverse dall'agente anche se le proprietà e le soluzioni ai problemi rimangono le stesse. In questa categoria potrebbero non venire rispettare le convenzioni di colorazione degli oggetti che non hanno una colorazione immutabile, come muri e rampe ad esempio, ma non le sfere, le quali hanno colori fissi.

7. **Modelli interni:** Viene testata la capacità dell'agente di memorizzare modelli interni dell'ambiente. In questi test la luce

dell'arena può essere spenta per un certo lasso di tempo e l'agente deve ricordare il layout dell'ambiente per poter navigare correttamente ed ottenere le ricompense. Sarà perciò richiesta una capacità di memorizzare informazioni sull'arena e prevedere gli stati futuri del mondo circostante.

8. **Permanenza degli oggetti:** Molti animali sembrano capire che quando un oggetto scompare dietro un altro esso esiste ancora. Esistono molte semplici interazioni che non sono possibili senza comprendere la permanenza degli oggetti e sarà interessante vedere come questo può essere codificato nei sistemi di intelligenza artificiale.

9. **Preferenze complesse:** Questa categoria verifica la capacità dell'agente di prendere decisioni più complesse per assicurarsi che ottenga la massima ricompensa possibile. Ciò potrebbe includere test con scelte che portano a diversi premi ottenibili in base a quale percorso si scegli di adottare.

10. **Ragionamento causale:** viene testata la capacità di pianificare in anticipo in modo che le conseguenze delle azioni siano considerate prima che vengano intraprese. Tutti i test in questa categoria sono stati superati da alcuni animali non umani e questi includono alcuni degli esempi più sorprendenti di intelligenza provenienti da tutto il regno animale.

Come si sarà notato le specifiche relative alle categorie presentate diventano via via più fumose procedendo dalle prime, più semplici, alle ultime, più complesse. Questo è dovuto proprio al fatto che le abilità cognitive che si tentano di verificare nelle ultime categorie hanno definizioni non banali e che possono includere moltissimi esempi.

3.3.5 *Valutazione*

L'agente, per essere valutato, va sottomesso in un'immagine docker alla piattaforma EvalAI[52], con le modalità spiegate nella documentazione della competizione[1].
Per tutta la durata della competizione i test che affronteranno gli agenti non verranno cambiati, essi sono 300 task equamente distribuiti nelle 10 dieci categorie. Un agente sottoposto verrà quindi valutato sequenzialmente su tutti e 300 i task, i quali hanno tutti una valutazione binaria, ovvero superato o meno. L'agente dovrà terminare tutti e 300 gli episodi entro 2 ore, altrimenti verrà interrotto e verranno considerati i punteggi ottenuti fino a quel momento. Una volta terminata la valutazione si riceverà un file di risultato, con il punteggio ottenuto in totale ed in ogni categoria.

1 https://github.com/beyretb/AnimalAI-Olympics/blob/master/documentation/submission.md

Per il primo mese sarà consentito un solo invio al giorno, mentre poi sarà possibile inviare 3 volte al giorno il proprio codice da testare.
Contestualmente su EvalAI una leaderboard si aggiornerà automaticamente in base ai risultati ottenuti dai vari team partecipanti.
L'accesso alla competizione è aperto a tutti, sia singolarmente che come team.
La competizione è stata divisa in due macro-fasi:

- **Dal 8 luglio 2019 al 1 settembre 2019:** in questa fase coloro che si troveranno nelle prime 20 posizioni della classifica riceveranno 500$ in crediti per Amazon AWS.
- **Fino al 1 novembre 2019:** in questa data la competizione finisce ufficialmente, ma la classifica non verrà stilata sulla base degli stessi test utilizzati durante tutta la competizione, bensì delle piccole variazioni, sia in termini di numero che di caratteristiche, dei task esistenti. Questo per evitare che la classifica sia il risultato circostanze fortuite o altri accorgimenti atti solo a massimizzare il punteggio. In questo caso i premi 3 in classifica verranno premiati rispettivamente con 7500, 5000 e 1500 euro. Saranno inoltre premiati con 200$ i partecipanti che hanno ottenuto il miglior punteggio in una delle categorie in gara.

4

MAPPATURA DELL'AMBIENTE

In questa sezione si discuterà delle possibili metodologie da mettere in campo per costruire una rappresentazione dell'ambiente in base alle esperienze dell'agente, nell'ottica di andare verso approcci di RL Model-Based, i quali potrebbero essere degli ottimi candidati per svolgere al meglio le sfide poste dalla competizione.
In letteratura questo problema di esplorazione e mapping è noto come **SLAM** (Simultaneous Localization and Mapping) e rappresenta una grande sfida nel mondo della robotica e non solo. Un robot per muoversi con precisione e svolgere i suoi compiti deve infatti avere una mappa accurata dell'ambiente in cui si muove, di conseguenza servirà anche localizzare accuratamente anche la posizione del robot stesso.
In questo campo di ricerca sono stati proposti svariati metodi, dapprima basate sull'applicazione di filtri, le tecniche di SLAM più utilizzate oggi si basano sull'utilizzo di grafi[14].
L'approccio che si utilizza per effettuare SLAM dipende strettamente anche da quale hardware (virtuale nel nostro caso) si ha a disposizione, la complessità del problema infatti cambia se si ha a disposizione un sensore LIDAR, una telecamera RGB-D o una semplice fotocamera monoculare come in questo caso.
Date le stringenti tempistiche della competizione questo capitolo deve essere visto come un indagine esplorativa sulle sfide poste dalla mappatura di un ambiente a partire da immagini monoculari. Si partirà quindi dalle motivazioni per cui si è scelto di indagare sui possibili meccanismi per la mappatura di ambienti e come questo potrebbe avere effetti sulla capacità di svolgere i test della competizione.
Si procederà poi mostrando com'è possibile costruire un dataset di immagini RGB-D a partire dall'ambiente della competizione per poi utilizzarle nell'addestramento supervisionato di un modello che stimi la distanza degli oggetti presenti in una scena, anche mostrando i risultati ottenuti.
Verranno infine presentate delle soluzioni euristiche al problema del mapping dell'ambiente e verrà spiegato anche perché si sono rese necessarie.

4.1 MOTIVAZIONI

Nel capito precedente sono state evidenziate le diverse categorie in cui sono divisi i test della competizione, alcune di esse sono abbastanza semplici e richiedono solo l'apprendimento di semplici associazioni segnale-ricompensa, ovvero se si vede una sfera verde o gialla è bene andarle incontro per raccoglierla ed ottenere una ricompensa.
Senz'altro quindi per avere successo nella prima categoria è sufficiente che l'agente possieda la capacità di andare nella direzione di stimoli positivi, già dalla seconda categoria viene introdotto un elemento non banale, ovvero le preferenze, fare bene in questa categoria significa disporre della capacità di valutare le dimensioni e la distanza di un oggetto, valutando l'ottenimento di quale ricompensa è preferibile perseguire.
A partire dalla terza categoria risulta indispensabile che l'agente possieda l'idea che dietro degli ostacoli potrebbe nascondersi del cibo, rendendo perciò necessaria un'esplorazione dell'arena.
Vista la durata molto spesso limitata degli episodi è però necessario che questa esplorazione sia cosciente e non puramente una serie di movimenti casuali, sarà infatti importante evitare di visitare delle porzioni dell'arena già visitate.
La quarta categoria, che magari può risultare a prima vista semplice visto che richiede di evitare semplicemente degli stimoli negativi, nasconde anch'essa delle sfide non banali. Bisogna infatti ricordare che l'agente dovrà comunque ottenere del cibo e perciò pur evitando le zone pericolose esso dovrà tenere ben presente dove si trova il suo obiettivo (se lo ha già visto).
Come evidenziato nei paragrafi precedenti precedenti quindi anche le categorie che sembrano più semplici nascondono in realtà una serie di sfide non banali, richiedendo una comprensione, almeno di base, della disposizione degli oggetti nell'arena.
Una comprensione molto maggiore dell'ambiente è invece necessaria in tutte le categorie successive, richiedendo una navigazione efficace dell'arena, in taluni casi addirittura mantenere un modello interno ed utilizzarlo per arrivare al proprio obiettivo senza poter utilizzare la vista.
Per fare questo l'agente dovrebbe quindi, a partire dalle singole immagini RGB ottenute ad ogni passo, ricostruire una sorta di mappa, magari 3D, dell'ambiente e sfruttarla per svolgere tutte le tipologie di task.
Se si avesse a disposizione una mappa così precisa la si potrebbe addirittura sfruttare per pianificare in anticipo tutte le azioni da compiere, valutando i possibili stati che il mondo assumerà in base alle azioni intraprese dall'agente.
Risulterà ormai chiaro perché una mappa potrebbe essere risolutiva per superare molte delle sfide poste dalla competizione, anche considerando che tutti gli animali, ed ancor di più gli uomini, dispongono di

un modello dell'ambiente che li circonda, modello che viene sfruttato continuamente nel perseguire i propri bisogni e desideri.
Questa esigenza però si scontra con problematiche tecniche non banali, bisogna infatti ricordare che l'agente è dotato di vista monoculare, mentre l'uomo e praticamente tutti gli animali sono dotati di due occhi, tramite i quali essi possono stimare la profondità degli oggetti nel proprio campo visivo. Questo può limitare molto la capacità di fare un mapping efficace dell'ambiente e richiederà che tutto il processo parta da immagini RGB.

Dato che per creare una mappa le informazioni sulla distanza degli oggetti dall'agente sono fondamentali, bisognerà necessariamente ricavare questa informazione in qualche altro modo. Una possibile via è quella di addestrare un modello che a partire da immagini monoculari riesca a ricavare le informazione relative alla profondità, come in [10] e [7]. Vista la specificità dell'ambiente della competizione e considerando che i modelli disponibili online a questo scopo sono stati addestrati su immagini reali, bisognerà addestrare un modello da zero. A questo scopo si è quindi deciso di procedere così:

- **Ottenere un training set di immagini RGB-D:** per effettuare un addestramento supervisionato sarà necessario disporre di un dataset che mappi delle immagini RGB provenienti dall'ambiente della competizione alle corrispettive RGB-D. Dato che gli organizzatori hanno fornito il codice con cui è stato sviluppato l'ambiente Unity, si potrebbe modificarlo aggiungendo una camera aggiuntiva, da sfruttare per ricevere immagini stereo. Ottenute queste immagini si potrà quindi stimare la distanza tramite opportune tecniche.

- **Addestrare il modello di stima profondità:** ottenuto il dataset si dovrà concepire un modello adatto alla stima della profondità.

- **Costruzione di mappe a partire da immagini RGB-D:** se i punti precedenti raggiungeranno risultati soddisfacenti si potrà sfruttare l'output del modello addestrato per costruire una mappa dell'ambiente.

4.2 OTTENERE UN TRAINING SET DI IMMAGINI RGB-D

In questa sezione verranno descritti i passi necessari ad ottenere il training set necessario per addestrare il modello di stima della profondità a partire da immagini monoculari. Si partirà però dalle basi teoriche dietro la visione stereo per poi mostrare come si è calata la teoria nel problema reale.

4.2.1 *Stereoscopia*

La visione stereoscopica, nell'ambito dell'informatica, si propone di, data una coppia di immagini stereo calibrate, ottenere l'informazione relativa alla profondità di ogni punto osservato.
L'idea di base è che comparando le informazioni proveniente da due punti, nel caso più semplice, sulla stessa linea orizzontale, è possibile confrontare la posizione dei punti nelle due immagini, sfruttando la diversa posizione per calcolare la distanza. Il risultato di questa comparazione viene detto **disparity map** e codifica la differenza in termini di coordinate orizzontali di punti corrispondenti. I valori in questa mappa di disparità sono inversamente proporzionali alla profondità della scena nella posizione corrispondente del pixel.

4.2.1.1 *Principio di base*

Si procederà ora nel descrivere matematicamente l'intuizione su cui si basa la stereoscopia. Come riferimento per la dimostrazione si utilizzi la figura 18.
Nella visione stereoscopica si hanno due immagini della stessa scena ma catturate da due diverse posizioni. Calando la situazione nel grafico 18, si ha che la luce dal punto A passa attraverso il foro stenopeico delle due camere in posizione B e D per essere proiettato sulle immagini nei punti E ed H.
La distanza fra i centri delle due lenti delle fotocamere è $BD = BC + CD$.
Come si può notare vi sono due coppie di triangoli simili:

- ACB e BFE
- ACD e DGH

Perciò la disparità del punto nelle due immagini varrà:

$$
\begin{gathered}
d = EF + GH \\
d = VF(\frac{EF}{BF} + \frac{GH}{BF} \\
d = BF(\frac{EF}{BF} + \frac{GH}{DG} \\
d = VF(\frac{BC + CD}{AC}) \\
d = BF\frac{BD}{AC} = \frac{k}{z} \\
dove\ k = BD\,BF,\ Z = AC
\end{gathered}
$$

La formula della disparità si può riscrivere anche come:

$$x - x' = \frac{B \cdot f}{z} \tag{20}$$

Con B detta baseline, ovvero distanza fra il centro delle due lenti, f è

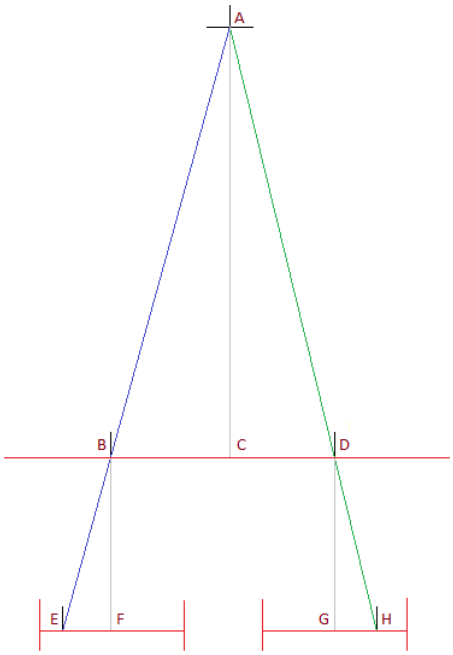


Figura 18: Principio base stereo vision. Tratta da [51]

la lunghezza focale (spazio tra centro ottico di una lente ed il punto di focalizzazione dei raggi entranti) e z è la distanza fra l'osservatore ed il punto.
Perciò da questa formula è possibile ricavare la distanza dell'oggetto data la disparità, ovvero:

$$z = \frac{B \cdot f}{x - x'} \tag{21}$$

Ovviamente questa formula vale soltanto se le due camere sono sullo stesso piano, se invece esse non sono coplanari allora sarà necessario eseguire la procedura della rettificazione delle immagini.
Il valore della baseline è molto importante nel risultato della stima della profondità, infatti se:

- **Baseline piccola:** maggiore possibilità di errori nella stima della profondità ma si riesce meglio a trovare le corrispondenze fra punti.
- **Baseline ampia:** la stima della profondità è più accurata ma è più difficile trovare i punti che fanno match, avendo più occlusioni.

Una visualizzazione intuitiva di quanto appena elencato è contenuta nell'immagine 19

4.2.1.2 *Fare matching fra i punti*

Nella sezione precedente si è dato per scontato che la posizione del punto reale nelle due due immagini fosse nota, in realtà questo è un punto

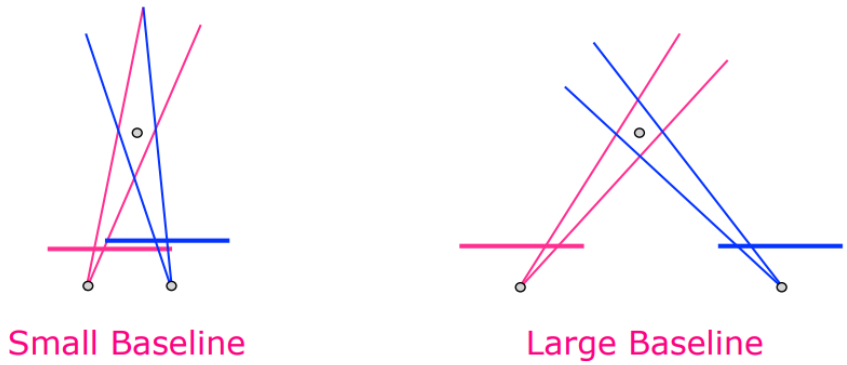


Figura 19: Effetto della baseline sulla stima della distanza. Tratta da [33]

critici nel processo di stereo visione. Infatti per calcolare la disparità dei punti bisogna prima trovare le corrispondenze fra di essi, quindi l'algoritmo più semplice che si può mettere in campo prevede che: per ogni finestra di dimensioni $(r \times r)$ con centro (x, y) sull'immagine di sinistra si cerchi, nell'immagine di destra, sullo stesso piano epipolare, il contenuto di una finestra $(r \times r)$ di punti più simile secondo una metrica scelta.
Scelte tipiche per questa metrica sono:

- Correlazione normalizzata
- SSD (Sum of Squared Differences) - somma delle differenze al quadrato

Dimensionare la finestra è un aspetto molto importante, in quanto se la finestra è piccola si hanno maggiori dettagli ma è il risultato finale è più rumoroso. Se invece si utilizzano finestre più grandi il risultato è meno rumoroso ma si ha meno dettaglio.

4.2.2 *Ottenere immagini stereo da Unity*

4.2.2.1 *Introduzione*

Come già descritto, l'ambiente in cui agisce l'agente è stato sviluppato in Unity, un motore grafico tra i più famosi, facendo uso del toolkit ML-Agents.
Nella competizione l'agente, per raccogliere le informazioni visive, è dotato di una telecamera posta al centro di esso, quindi come già discusso non può direttamente raccogliere immagini stereoscopiche. Fortunatamente gli organizzatori della competizione hanno messo a disposizione il codice sorgente dell'ambiente sviluppato per la competizione, dando quindi la possibilità di cambiarlo per i propri scopi in fase di addestramento.
In questo caso l'idea è stata quella di aggiungere altre due camere a quella già presente, in modo da far percepire all'agente sia l'immagine

catturata da quella centrale (ovvero quella ufficiale, usata anche nella competizione) sia le due camere addizionali, una posta a sinistra e l'altra a destra di quella centrale.
Averle disposte in questo modo facilità anche la stima della profondità perché non richiede di eseguire il processo di rettificazione delle immagini.
Una volta che si ha a disposizione questa variazione dell'ambiente originale, si può prendere un modello di agente precedentemente addestrato e farlo girare su alcuni scenari preparati in precedenza. Ad ogni passo temporale l'agente riceverà quindi dall'ambiente Unity 3 immagini (la sua observation), le quali andranno a costituire un dataset di immagini monoculari collegate alle corrispondenti immagini stereo.
Si è pensato di utilizzare un modello già addestrato perché il dataset risultante dovrebbe contenere immagini più significative rispetto a quelle ottenute da una policy random.

4.2.2.2 *Aggiunta delle camere*

Grazie alla grande flessibilità di Unity, è stato possibile aggiungere due ulteriori fotocamere all'oggetto rappresentante l'agente. Per modificare o aggiungere elementi della scena base dell'environment Unity bisogna agire sul **prefab** dell'arena.I prefabs (abbreviazione che sta per 'prefabbricati') sono dei gameObject prefabbricati . I prefabs sono utili ogni volta che un oggetto deve comparire più di una volta, e deve avere caratteristiche uguali o simili a quella di un altro, ma non per questo sempre identiche. Ogni volta che un prefab viene messo in scena, esso rappresenta una specifica istanza del prefab stesso.
Nel caso della competizione esso viene poi arricchito da gli oggetti specificati nel file di configurazione.

Si è quindi proceduto a creare due copie di camere: **AgentCamLeft** ed **AgentCamRight**, con le caratteristiche evidenziate nella figura 20. Delle informazioni riportate nella figura le più importanti per stimare la distanza sono:

- **La posizione sull'asse X**: si è deciso di porre le camere ad una distanza di 15 cm dal centro dell'agente, quindi la baseline B è di 30 cm. La scelta di questo parametro è stata empirica, in base ai risultati ottenuti nella successiva stima della profondità.

- **La lunghezza focale:** è una proprietà della lente e vale 22.16 mm.

A questo punto per rendere effettive ed utilizzabili le camere aggiuntive sono necessari altri due passi.
Il primo consiste nello specificare nella sezione apposita relativa al Training Agent (un componente specifico di ML-Agents) tutte le camere che

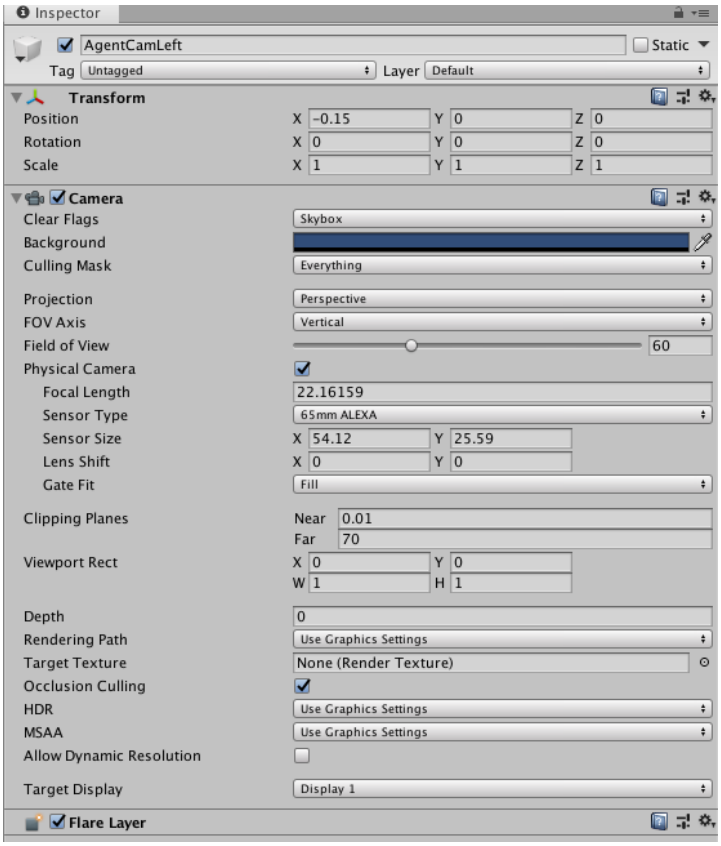


Figura 20: Configurazione della fotocamera sinistra dell'agente.

esso deve sfruttare, come mostrato in figura 21. L'ultimo passo prevede invece di modificare lo script che specifica lo spazio delle osservazioni e delle azioni dell'agente, nel gergo di ML-Agents esso è detto Brain. Si è perciò proceduto ad aggiungere le informazioni provenienti dalle fotocamere aggiuntive, rendendo il vettore contenente le observation un vettore di forma $(3, 84, 84)$.
La configurazione di questo componente è evidenziata nella figura 22. A titolo esemplificativo infine un esempio di immagini catturate ad un certo istante temporale dall'agente, si può notare sopratutto dalla diversa posizione delle righe del pavimento la differente posizione orizzontale delle camere.

4.2.3 *Stima della depth da immagini stereo*

Avendo ora a disposizione un dataset di immagini stereo si potrà stimare la distanza degli oggetti nelle varie immagini raccolte, sfruttando i principi della stereoscopia, già introdotti nella sezione 4.2.1. In questa fase si ricaveranno quindi le informazioni relative alla distanza di ogni punto facente parte dell'immagine, creando un dataset di immagini e corrispondenti valori di profondità.
Per ottenere la profondità a partire dalle immagini stereo ci si è basati

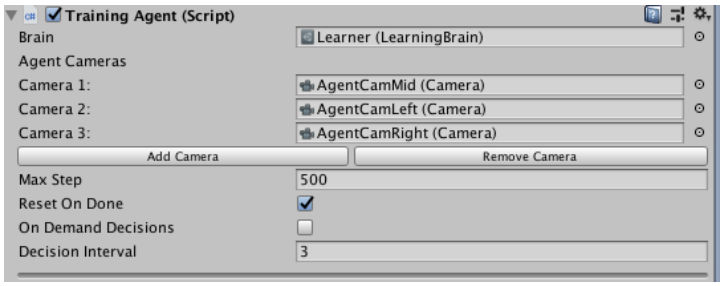


Figura 21: Configurazione del Training Agent.

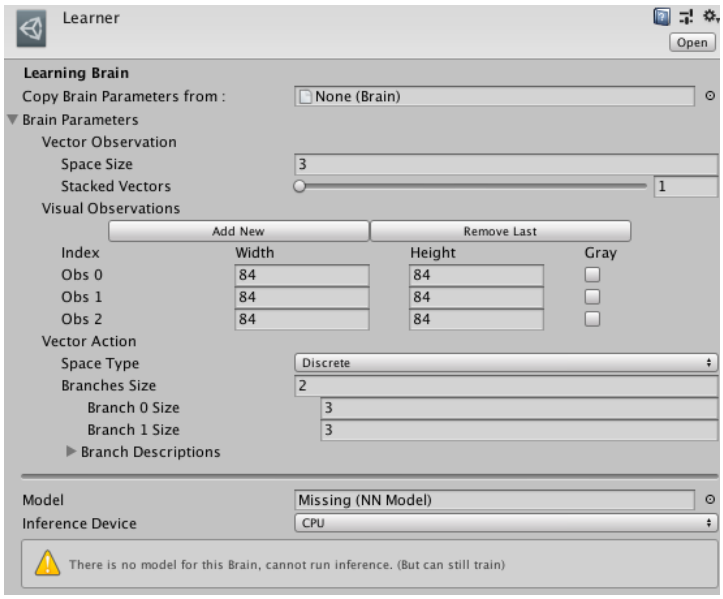


Figura 22: Configurazione del componente Brain dell'agente.

sulle implementazioni fornite da librerie di visione artificiale molto conosciute, come OpenCV [4].

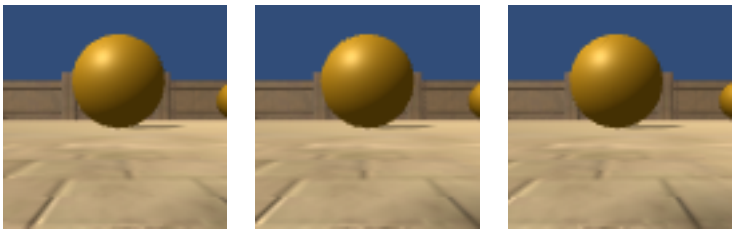
4.2.3.1 *Implementazione*

Nello specifico la soluzione adottata si divide in tre fasi:

- **Calcolo delle disparità ed applicazione filtro:** a partire dalle due immagini stereo si l'algoritmo SGBM fornito nelle libreria OpenCV. Esso è una variante[1] dell'algoritmo Semi-Global Matching [13]. Contestualmente al calcolo della disparità è stato applicato il filtro Weighted Least Squares[2], solitamente utilizzato per migliorare i risultati ottenuti, sopratutto in caso di occlusioni ed aree uniformi. Di seguito la porzione di codice che descrive il calcolo della disparità:

1 Le differenze sono ritrovabili all'indirizzo https://docs.opencv.org/3.4/d2/d85/classcv_1_1StereoSGBM.html

2 Descritto nella documentazione: https://docs.opencv.org/3.4/db/d72/classcv_1_1ximgproc_1_1DisparityFilter.html



(a) *Immagine stereo sinistra* (b) *Immagine centrale* (c) *Immagine stereo sinistra*

Figura 23: Esempio delle tre immagini acquisite in certo istante temporale dall'agente.

```
    fx = 22.16159  #mm
    baseline = 300  #mm
    disparities = 16
    block = 7
    units = 1
    resolution = 84
    window_size = 7
    left_matcher = cv2.StereoSGBM_create(
        minDisparity=-1,
        numDisparities=1* 16,
        blockSize=window_size,
        P1=8 * 3 * window_size,
        P2=32 * 3 * window_size,
        disp12MaxDiff=12,
        uniquenessRatio=10,
        speckleWindowSize=50,
        speckleRange=32,
        preFilterCap=63,
        mode=cv2.STEREO_SGBM_MODE_SGBM_3WAY)

    right_matcher = cv2.ximgproc.createRightMatcher(
left_matcher)
    #Parametri del filtro
    lmbda = 80000
    sigma = 1.3
    visual_multiplier = 6
    wls_filter = cv2.ximgproc.createDisparityWLSFilter
(matcher_left=left_matcher)
    wls_filter.setLambda(lmbda)
    wls_filter.setSigmaColor(sigma)
    displ = left_matcher.compute(left, right)
    dispr = right_matcher.compute(right, left)
    displ = np.int16(displ)
    dispr = np.int16(dispr)

    filteredImg = wls_filter.filter(displ, left, None,
     dispr)
    filteredImg = cv2.normalize(src=filteredImg, dst=
filteredImg, beta=0, alpha=255,
                                norm_type=cv2.NORM_MINMAX)
    filteredImg = np.uint8(filteredImg)

```

- **Calcolo delle distanze a partire dalle disparità:** Ottenuti i valori delle disparità si potrà, come mostrato nella sezione 4.2.1, calcolare la depth dei punti, ovviamente utilizzando i valori mostrati nella sezione 4.2.2.2, relativi a baseline e lunghezza focale.

```
1    disparity = filteredImg
2    valid_pixels = disparity > 0
3    depth = np.zeros(shape=(resolution, resolution)).
     astype("uint8")
4    depth[valid_pixels] = (fx * baseline) / (units *
     disparity[valid_pixels])
5
```

- **Calcolo di media spaziale e temporale sulla distanza:** Considerato che molto spesso i risultati provenienti da immagini stereo possono essere molto rumorosi, si è deciso di adottare alcuni accorgimenti per rendere meno rumorose e più "smooth" le immagini. Innanzitutto dopo aver eseguito la stima della profondità si esegue una convoluzione usando come kernel una matrice di dimensione (w, w) con tutti i valori posti a $1/(w \times w)$ e mantenendo le stesse dimensioni di partenza, calcolando di fatto la media nella finestra considerata:

```
1    kernel = np.ones((3, 3), np.float32) / 9
2    depth = cv2.filter2D(depth, -1, kernel)
3
```

  Il procedimento appena illustrato riguarda la media dei punti nello spazio, mentre per quanto riguarda la media nel tempo si è deciso di considerare i dati della profondità di tre in tre, calcolando la media su di essi ed associando ad ogni immagine di partenza come profondità proprio la media calcolata.
  Infine i valori calcolati vengono normalizzati tramite la formula $\frac{X - x_{min}}{x_{max} - x_{min}}$ considerando come valore minimo 0 e come valore massimo 255.

4.2.3.2 *Risultati*

Grazie ad i passi descritti in tutta questa sezione è stato possibile creare un dataset comprendente circa 70000 immagini monoculari con corrispondenti valori di profondità. Per raccogliere queste immagini sono stati utilizzati 4 diversi modelli di agente, eseguiti su 120 configurazioni di test ideate nell'ambito dello sviluppo di questo elaborato di tesi.
Tuttavia i risultati di questa fase purtroppo non sono stati particolarmente brillanti, pur non avendo i mezzi per valutare le distanze effettive e calcolare perciò l'errore nella stima, qualitativamente, osservando le immagini risultanti non si può ritenere davvero accurata la stima.
A titolo d'esempio vengono di seguito riportate alcune sequenze di frame catturati dall'agente e la relativa stima della distanza tramite stereoscopia. L'immagine risultante, seppure abbastanza omogenea e

non troppo rumorosa, manca di dettaglio, come si nota sopratutto nella figura 24. Il risultato non soddisfacente potrebbe essere dovuto alla natura dell'ambiente in cui sono state catturate le immagini, infatti trattandosi di un ambiente sintetico, in cui le ombre sono minime e le superfici molto omogenee, l'algoritmo di matching fra punti corrispondenti potrebbe non trovare le giuste corrispondenze.
Ovviamente i parametri dell'algoritmo si posso calibrare per ottenere risultati diversi, magari aumentando la numerosità dei dettagli anche se ciò significa aumentare la rumorosità dell'immagine, in questo caso però è difficile calibrare questi parametri in quanto il risultato dipende anche dal tipo di scena che si ha davanti. Infatti l'agente mentre si muove può incorrere ad oggetti davvero molto vicini ma anche oggetti molto lontani ed è complesso trovare un valore che vada bene per tutte le casistiche.
Disporre di un training set di bassa qualità renderà quindi ancora più complesso il task della stima della profondità a partire da immagini monoculari che si vuole compiere.

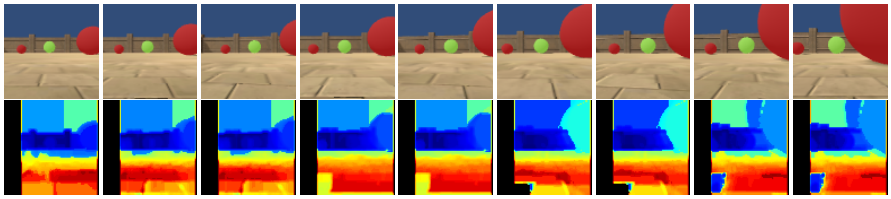


Figura 24: Esempio di immagine monoculare originale e stima della profondità in un arena vuota con una sola piccola sfera verde in lontananza

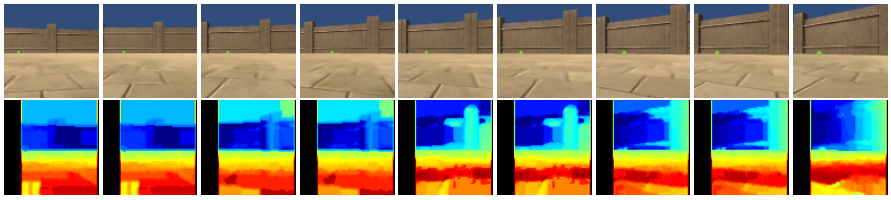


Figura 25: Esempio di immagine monoculare originale e stima della profondità in un arena con due sfere rosse ed una verde.

4.3 STIMARE DEPTH DA IMMAGINI MONOCULARI

Avendo ora costruito il dataset formato da { X: immagini RGB monoculari, Y: profondità dei punti delle immagini } si può addestrare uno specifico modello per stimare i valori di profondità a partire dalla immagini RGB monoculari.
Nella scelta del modello da utilizzare si è optato per architetture non troppo complesse, che dessero la possibilità di essere addestrare in un tempo ragionevole e che garantissero una velocità d'inferenza compatibile con l'utilizzo in seno all'agente. Va infatti ricordato che l'obiettivo

finale è incorporare nell'agente un modello che stimi la distanza dei punti nel suo campo visivo e che utilizzi queste informazioni per costruire una mappa dell'arena.
Dato che l'implementazione di un modello efficace, per task di visione artificiale complessi come questo, richiede molto tempo ed esperienza, si è deciso di affidarsi ad architetture già consolidate nel campo della ricerca, riadattandole allo scopo prefissato.

Un task che può, per certi versi, essere considerato simile alla depth estimation è la segmentazione. Essa, come intuibile dal nome, ha come scopo quello suddividere l'immagini in porzioni distinte, andando ad individuare degli insieme di pixel che hanno caratteristiche comuni. Un esempio classico è quello di individuare dove si trova un certo oggetto nell'immagine ed associare ad ogni pixel di esso un etichetta comune.
Il significato della stima della profondità non è così lontano, infatti i punti alla stessa distanza dall'osservatore hanno sicuramente delle caratteristiche in comune, rendendo possibile individuare quali sono più vicini e quali più lontani.
I due task vengono addirittura combinati in taluni casi [27, 26], permettendo di ottenere delle informazioni molto più significative rispetto ai singoli approcci. In questo caso tuttavia si è deciso di eseguire soltanto la stima della profondità perché l'obiettivo è di costruire una mappa dell'ambiente, sarebbe sicuramente molto interessante verifica l'efficacia di addestrare un algoritmo di reinforcement learning con l'output dei metodi sopra citati.
La scelta del modello è caduta su una architettura[45] proposta nei tutorial del popolare framework Tensorflow per fare segmentazione di immagini. L'Architettura scelta è stata quindi riadattata per utilizzarla nello stimare la profondità. Verranno quindi di seguito introdotte alcune nozioni di base riguardo al modello scelto, per poi riportare le modifiche apportate e le scelte intraprese, concludendo con un analisi dei risultati ottenuti.

4.3.1 *I modelli U-Net, Pix2Pix e la segmentazione d'immagini*

4.3.1.1 *U-net*

La rete U-net, proposta da Ronneberger et al., è un evoluzione della rete "fully convolutional" [18], ed è nata allo scopo di segmentare immagini biomedicali con maggiore accuratezza ed efficienza, ottenendo risultati migliori pur necessitando di minori dati per l'addestramento.
Come si evince dalla figura 26 l'architettura è simmetrica (mentre la FCN non lo era) ed esistono delle connessioni dirette fra i livelli simmetrici (cosiddette "skip connection"). L'architettura può essere divisa concettualmente in 3 parti:

- La parte di downsampling: essa è composta da 4 blocchi a loro

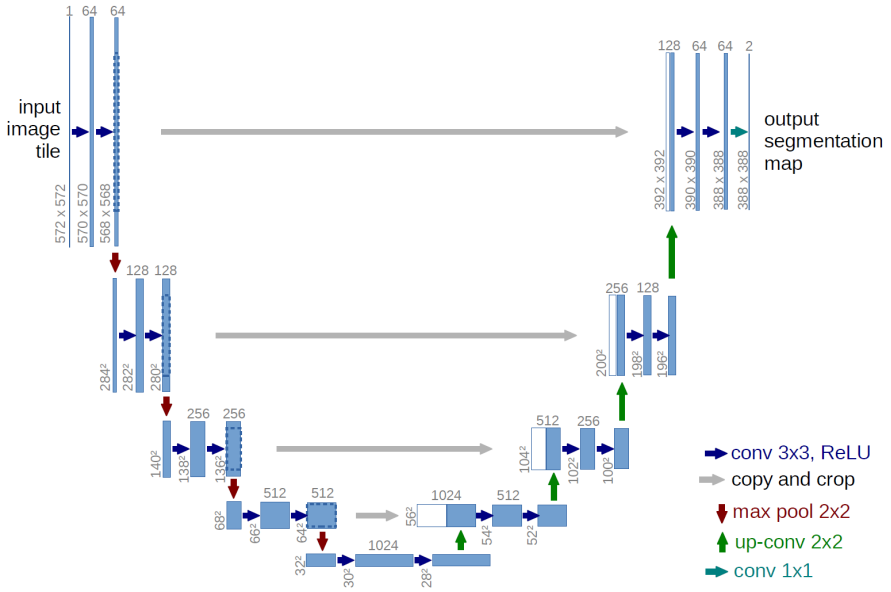


Figura 26: Architettura della U-net. Tratta da [34]

volta contenenti due strati di convoluzione ed uno di max pooling. Dopo ogni pooling il numero di feature raddoppia, passando dalle 64 di partenza alle 1024 in ingresso alla porzione centrale. L'obiettivo di aumentare il numero di feature ma comprimendo l'immagine originale è di catturare al meglio il contesto delle immagini e tentare di apprendere la struttura implicita in esse

- Il collo di bottiglia: è la parte di collegamento fra la porzione di upsampling e quella di downsampling, è composta di due soli livelli di convoluzione.

- La parte di upsampling: viene considerata la parte più innovativa, anch'essa è composta da quattro blocchi, ma in questo caso ad ogni passo viene fatto un upsampling della feature map, seguita da uno strato di convoluzione 2x2 (definita up-convolution) la quale dimezza il numero di canali di feature. Contestualmente viene concatenata la feature map proveniente dal livello corrispondente della prima parte della rete, e poi applicati due livelli di convoluzione 3x3, ognuno di essi terminante con uno strato ReLu[3]. Il vantaggio di avere queste connessioni dirette fra blocchi simmetrici è che ciò garantirà che le feature apprese durante la fase di downsampling verranno utilizzate per ricostruire l'immagine nella fase di upsampling.

4.3.1.2 *Pix2Pix*

Pix2Pix[15] è il nome che prende un lavoro davvero interessante proposto da Isola et al., nel quale vengono utilizzate le conditional Generative

3 Funzione di attivazione REctifier Linear Unit

Adversarial Networks (cGAN), per eseguire una traduzione da immagine ad immagine, come negli esempi nella figura 27, trasformando un soggetto composto da soli bordi in una sua versione colorata, o il disegno di una facciata di un palazzo in una sua rappresentazione realistica.
Ad alto livello l'approccio si basa su due principali componenti, il **Generator** ed il **Discriminator**, compito del primo è di applicare qualche tipo di trasformazione all'immagine di input per ottenere una certa immagine di output. Il secondo invece si occupa di confrontare l'immagine di input ad un'immagine di cui non conosce la provenienza e deve provare a capire se essa proviene dal Generator oppure no. Il punto principale è che non viene specificata una funzione di costo specifica per il task di trasformazione, bensì è il discriminator che implicitamente definisce una funzione di loss per il generator. Infatti i pesi del discriminator vengono aggiustati in base all'errore di classificazione tra le coppie input/output ed input/target, ma i pesi del generator vengono aggiustati sia sia sulla base dell'output del discriminator sia rispetto alla differenza tra output e target. Quindi di fatto si sta calcolando il gradiente anche attraverso il risultato del discriminator spingendo di fatto il primo a cercare di battere il secondo.
L'architettura del generator è infine composta da una U-net

4.3.1.3 *Segmentazione di immagini*

L'approccio adottato per la segmentazioni di immagini riportato da Tensorflow[45] si basa una versione modificata di una U-Net[34]. Questa tipologia di reti è composta due parti principali, la prima detta encoder (o downsampler) e la seconda detta decoder (o upsampler). Talvolta può essere molto vantaggioso utilizzare dei modelli pre-addestrati come encoder, in modo da ridurre il numero di parametri da addestrare e da poter apprendere delle feature più robuste e generali. In questo caso viene adottato il modello MobileNetV2[35], il cui obiettivo è proprio di fornire un modello leggero e performante. Per quanto riguarda la parte del decoder invece utilizzata la porzione di upsample di Pix2Pix[4]. Di seguito il codice adottato per la creazione del modello, tratto da [45]:

```
    base_model = tf.keras.applications.MobileNetV2(
    input_shape=[128, 128, 3], include_top=False)
    # Use the activations of these layers
    layer_names = [
        'block_1_expand_relu',   # 64x64
        'block_3_expand_relu',   # 32x32
        'block_6_expand_relu',   # 16x16
        'block_13_expand_relu',  # 8x8
        'block_16_project',      # 4x4
    ]
    layers = [base_model.get_layer(name).output for name in
     layer_names]

```

4 Nell'implementazione fornita da Tensorflow https://github.com/tensorflow/examples/blob/master/tensorflow_examples/models/pix2pix/pix2pix.py

Figura 27: Alcuni esempi applicativi dell'approccio Pix2Pix. Tratta da [15]

```
    # Create the feature extraction model
    down_stack = tf.keras.Model(inputs=base_model.input,
    outputs=layers)

    down_stack.trainable = False
    up_stack = [
        pix2pix.upsample(512, 3),  # 4x4 -> 8x8
        pix2pix.upsample(256, 3),  # 8x8 -> 16x16
        pix2pix.upsample(128, 3),  # 16x16 -> 32x32
        pix2pix.upsample(64, 3),   # 32x32 -> 64x64
    ]
    def unet_model(output_channels):
     # This is the last layer of the model
      last = tf.keras.layers.Conv2DTranspose(
          output_channels, 3, strides=2,
          padding='same', activation='softmax')  #64x64 ->
    128x128

      inputs = tf.keras.layers.Input(shape=[128, 128, 3])
      x = inputs

      # Downsampling through the model
      skips = down_stack(x)
```

```
33      x = skips[-1]
34      skips = reversed(skips[:-1])
35
36      # Upsampling and establishing the skip connections
37      for up, skip in zip(up_stack, skips):
38        x = up(x)
39        concat = tf.keras.layers.Concatenate()
40        x = concat([x, skip])
41
42      x = last(x)
43
44      return tf.keras.Model(inputs=inputs, outputs=x)
45
46    model = unet_model(OUTPUT_CHANNELS)
47    model.compile(optimizer='adam', loss='
      sparse_categorical_crossentropy',
48                  metrics=['accuracy'])
49
```

Si noti che l'ultimo livello l'output è composto da tre canali, dato che per il problema in questione esistono tre possibili etichette per ogni pixel, il che equivale, di fatto a fare una classificazione su tre classi.

4.3.2 *Adattamento del modello alla depth estimation*

Il modello appena presentato può essere abbastanza facilmente trasformato per eseguire la stima della profondità, in particolare quindi bisognerà variare l'ultimo livello della rete, che in questo caso non avrà tre canali, bensì un solo canale, e come funzione di attivazione invece di softmax verrà utilizzata la funzione sigmoide. Il motivo della scelta di questa funzione di attivazione è dettato dal fatto che i valori di profondità sono stati normalizzati tra 0 ed 1 e la sigmoide produce proprio valori compresi fra 0 ed 1.
Va anche considerato che le immagini di partenza sono più piccole di quelle utilizzate nel codice precedente, dovrà perciò essere utilizzate un modello di encoder pre-addestrato con delle dimensioni diverse, senza riportare nuovamente il codice è possibile visualizzare l'architettura del modello risultante nella figura 28. Dato che il modello di encoder pre-addestrato era disponibile per immagini almeno di dimensioni 96x96 è stato necessario aggiungere un padding alle immagini 84x84 in modo da raggiungere questa dimensione. Apposite funzioni per la costruzione di depth map a partire dalle stime di profondità relative alle immagini sono stati realizzate, sopratutto per disporre di un confronto visivo tra il risultato atteso e quello ottenuto. Ovviamente al fine addestrare il modello sono state definite apposite funzioni per il caricamento dei dati in modo efficiente, utilizzando le utility di tensorflow.

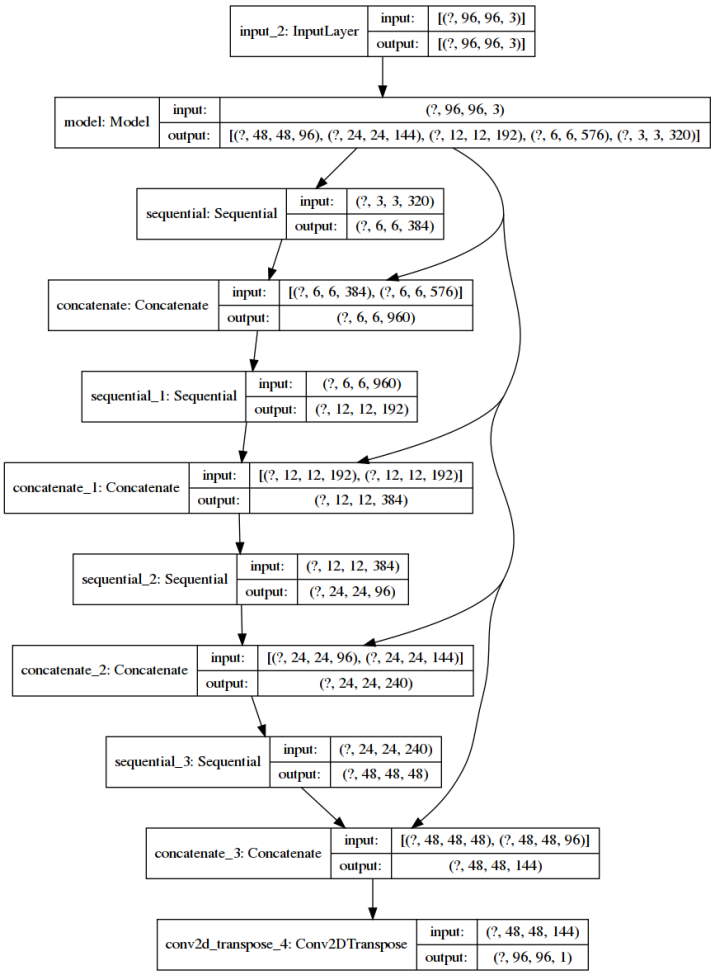


Figura 28: Variante di U-Net per stimare la profondità a partire da un immagine RGB

4.3.3 *Funzione di loss adottata*

Ovviamente la funzione di loss da adottare nell'addestramento non poteva essere la stessa usata per la segmentazione (ovvero la sparse categorical cross-entropy), infatti si è ritenuto vantaggioso utilizzare una funzione specifica per questo task. Riguardo alla depth estimation tramite addestramento supervisionato un ottima proposta è quella avanzata da Eigen and Fergus. Nel lavoro[7] si propone, per confrontare i valori stimati D e quelli veri D^* di utilizzare la seguente funzione:

$$L_{depth}(D, D^*) = \frac{1}{n}\sum_i d_i^2 - \frac{1}{2n^2}\Big(\sum_i d_i\Big)^2 + \frac{1}{n}\sum_i[(\nabla_x d_i)^2 + (\nabla_y d_i)^2] \tag{22}$$

con $d = D - D^*$.
L'equazione 22 è essenzialmente composta da tre termini:

- Il primo, ovvero $\frac{1}{n}\sum_i d_i^2$ è una classica distanza L2 (euclidea)

- Il secondo, ovvero $-\frac{1}{2n^2}\left(\sum_i d_i\right)^2$, ha lo scopo di diminuire la loss, che considerando la sua natura pixelwise potrebbe essere molto alta anche se le immagini sono piuttosto simili.

- Il terzo, ovvero $\frac{1}{n}\sum_i[(\nabla_x d_i)^2 + (\nabla_y d_i)^2]$, è un matching term sul gradiente, il cui scopo è quello di aiutare a mantenere la struttura generale dell'immagine e non solo la similarità tra pixel.

4.3.4 *Analisi dei risultati*

Il modello descritto in questa sezione è stato quindi addestrato su un dataset di 73995 immagini utilizzando come batch size 64 e compiendo 163 epoche di addestramento. La loss finale era di 38, un valore si è abbassato molto durante l'addestramento, partendo da circa 10000 come valore iniziale. Purtroppo però, come si evince dagli esempi riportati nella figura 29 i risultati non sono soddisfacenti. Le depth map prodotte sono totalmente caotiche, probabilmente a causa della bassa qualità del dataset di partenza. Durante le varie epoche di addestramento pur scendendo la loss la qualità delle depth map non è migliorata, ciò potrebbe essere chiaro indice di un dataset con informazioni incoerenti tra loro, portando a risultati scadenti.
Già di per se la stima della profondità a partire da immagini monoculari è un task abbastanza complesso e sicuramente i seguenti vincoli hanno incrementato ancor di più la difficoltà dello stesso:

- Necessità di un modello leggero e rapido per non appesantire il processo decisionale dell'agente. Si ricordi infatti che nella competizione il codice sottomesso ha un tempo limitato a 2 ore per terminare i 300 test.

- La sinteticità dell'ambiente Unity e la presenza di pochi dettagli ed ombre non ha permesso di formare un dataset di immagini RGB-D di qualità.

Dato che la via della stima della profondità non ha dato risultanti soddisfacenti si è deciso di non utilizzare il modello discusso in questa sezione per affrontare i test della competizione.

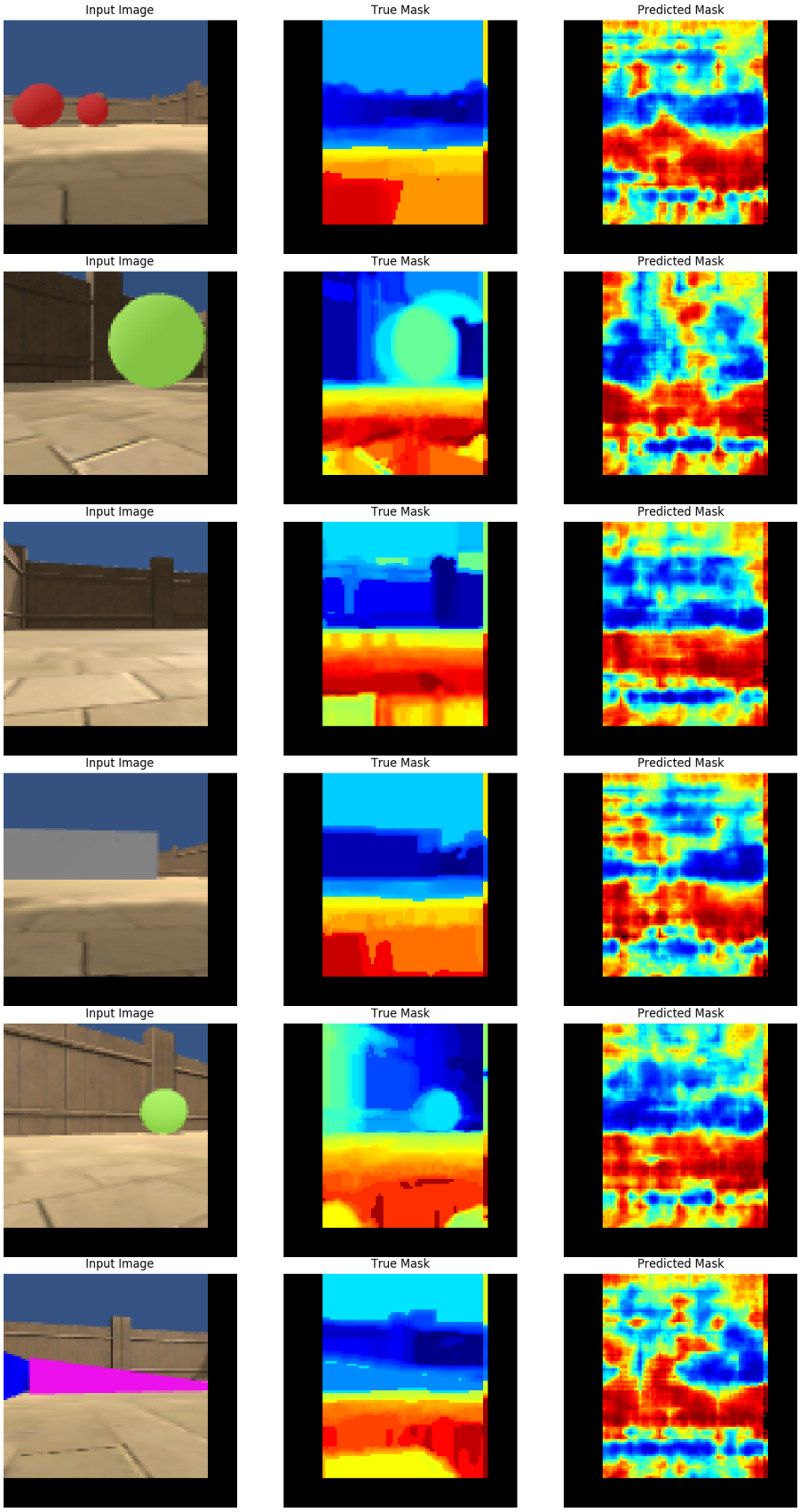


Figura 29: Alcuni esempi di depth estimation tramite l'approccio descritto. L'immagine di sinistra è quella di input, quella al centro è la depth mask target e quella a destra è quella prodotta dal modello.

4.4 SOLUZIONI EURISTICHE

Come si è evidenziato nella sezione precedente, l'approccio adottato per stimare la profondità dei vari elementi presenti in un immagine monoculare non ha dato i risultati sperati, impedendo quindi di creare una mappa dell'ambiente da sfruttare nell'eseguire i vari scenari della competizione.
Dato che disporre di una rappresentazione interna dello stato corrente dell'arena poteva risultare particolarmente vantaggioso, si è ritenuto che anche una rappresentazione più sommaria, ottenuta con metodi euristici, poteva essere realizzata.
L'approccio che verrà descritto si basa sia sul tracciamento del movimento dell'agente, ovviamente tenendo conto del fatto che non si è a conoscenza della posizione e dell'orientamento iniziale, sia su dei calcoli relativi al cono di visione dell'agente.
Il problema del tracciamento di movimento dell'agente in letteratura è noto come **odometria**[1], essa si basa sul principio che quando il robot si sposta per un certo periodo di tempo, la sua nuova posizione può essere determinata mediante l'integrazione della velocità del robot nel periodo del suo movimento, in modo da ottenere così la distanza percorsa. Se il robot cambia direzione mentre si muove, è necessario invece applicare la trigonometria per calcolare la nuova posizione.
L'odometria, sopratutto nei casi reali, è però soggetta a errori causati dall'incertezza delle misure provenienti dai componenti del robot e dalle irregolarità della superficie. Essendo il nostro caso di applicazione un ambiente virtuale, i difetti di questo approccio saranno molto meno marcati e l'unico principale problema sarà costituito dal non avere a disposizione la posizione e l'orientamento iniziale dell'agente.

4.4.1 *Tracciamento del movimento dell'agente*

4.4.1.1 *Dimensione della mappa*

L'idea è quella di costruire una mappa "agente-centrica", in quanto non avendo a disposizione la posizione e l'orientamento iniziale si utilizza una dimensione tale che anche nel caso peggiore tutto ciò che viene rilevato viene riportato sulla mappa.
Essendo l'arena di dimensioni 40x40, il caso peggiore è che l'agente si trovi in un angolo di essa ed orientato verso il centro percorra tutta la diagonale, in questo caso percorrerà 56 metri e perciò nella mappa ci dovrà essere spazio a sufficienza per memorizzare questo movimento. Non potendo intuire da dove l'agente partirà bisogna perciò utilizzare una mappa di dimensioni **112x112** e porre il punto di partenza dell'agente alla posizione (56,56) ed il suo angolo di orientamento iniziale a 0°. Così facendo qualsiasi situazione può essere mappata.

4.4.1.2 *Tracciamento del movimento*

Per tracciare i movimenti dell'agente, ovviamente basandosi sulle informazioni a disposizione, richiede di mantenere 4 informazioni:

- **L'orientamento (conosciuto) dell'agente:** come già anticipato si considera di partire da un angolo di orientamento uguale a 0°, che nel sistema di riferimento della competizione equivale all'agente che posizionato a z = 0 guarda nella direzione dell'asse z. In base all'azione eseguita ad ogni passo temporale si andrà ad aggiornare l'orientamento aggiungendo un angolo specificato con la seguente modalità: azione 0 = 0°, azione 1 = 6°, azione 2 = -6°. Per i dettagli riguardo al modello di movimento dell'agente si faccia riferimento alla sezione 3.3.2.

- La posizione sull'asse X,Z,Y: come già detto la posizione iniziale è posta a (56,56,0).

Per calcolare la distanza percorsa ad ogni step si utilizzano delle formule trigonometriche basandosi sui valori di velocità percepiti dall'agente. Si ricordi infatti che le velocità in questione hanno come sistema di riferimento l'agente stesso, per cui l'asse z (ovvero la terza componente, nei sistemi di riferimento classici essa sarebbe in realtà la y) punta sempre nella direzione frontale dell'agente. Questo aspetto va tenuto in considerazione quando si calcola la nuova posizione, in quanto va considerato anche l'angolo di orientamento dell'agente per poter determinare le componenti dello spostamento rispetto al sistema di riferimento dell'arena.
Si procede quindi calcolando la dimensione dei vettori spostamento sui tre assi, seguendo quando descritto nella sezione 3.3.2.

$$
\begin{aligned}
x_{\Delta distance} &= 0.0595 * x_{speed} \\
y_{\Delta distance} &= 0.0595 * y_{speed} \\
z_{\Delta distance} &= 0.0595 * z_{speed}
\end{aligned}
$$

Ottenute le dimensione dei vettori spostamento sugli assi x, y e z rispetto all'agente, si possono calcolare le componenti x e z (rispetto al sistema di riferimento dell'arena) dello spostamento effettuato dall'agente:

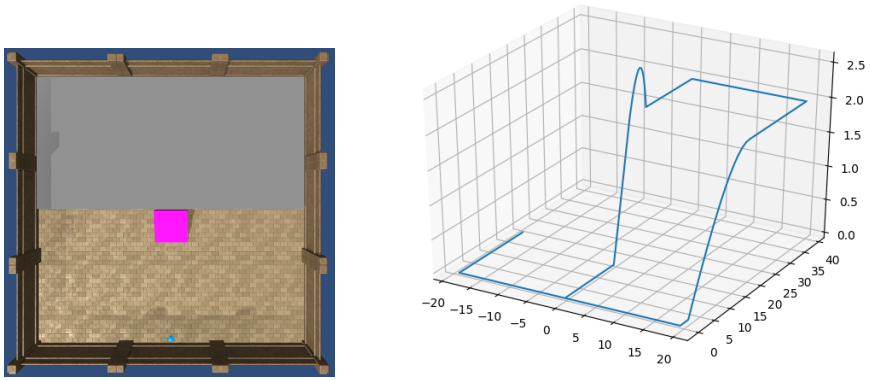
$$
\begin{aligned}
z_z &= z_{\Delta distance} \times \cos(\alpha) \\
z_x &= z_{\Delta distance} \times \sin(\alpha) \\
x_z &= x_{\Delta distance} \times \cos(\alpha + 90°) \\
x_x &= x_{\Delta distance} \times \sin(\alpha + 90°)
\end{aligned}
$$

Con α angolo di orientamento dell'agente. Il secondo angolo che viene utilizzato equivale all'angolo dell'agente con l'aggiunta di 90° in quanto l'asse x è perpendicolare all'asse z. In questo caso si sommano 90°

invece di sottrarli perché nel sistema di riferimento dell'arena gli angolo crescono andando da sinistra verso destra. La nuova posizione dell'agente è quindi data da:

$$x_{position} \mathrel{+}= z_x + x_x$$
$$y_{position} \mathrel{+}= y_{\Delta distance}$$
$$z_{position} \mathrel{+}= z_z + x_z$$

Un esempio di tracciamento della posizione dell'agente è evidenziato in figura 30, in questo caso nel grafico a titolo esemplificativo sono riportate gli spostamenti invece della mappa vera e propria, al fine di mostrare con maggiore chiarezza quanto appena descritto.
In questo caso l'agente è partito al centro rispetto all'asse x ed in basso dell'asse z, esso è andata avanti fino ad incontrare una rampa che lo ha quindi fatto salire su di una piattaforma, per poi sterzare a destra una volta arrivato in fondo all'arena, scendere dalla piattaforma e costeggiare i muri dell'arena. Il risultato è molto fedele, pur considerando che il



(a) *Situazione iniziale dell'arena*

(b) *Grafico delle posizioni dell'agente*

Figura 30: Esempio di grafico delle posizioni dell'agente con immagine dall'altro dell'arena.

tracciamento si può basare solo sulle informazioni acquisite durante l'episodio, in quanto le informazioni iniziali non sono note.

4.4.2 *Utilizzare coni di visione per stimare distanza*

Conoscere gli spostamenti dell'agente è il primo passo necessario per poter costruire una rappresentazione euristica dell'arena. Il passo successivo è quello di rappresentare gli oggetti contenuti in essa, questo può essere fatto considerando quello che è il campo visivo dell'agente. L'agente ha infatti un angolo di visione di 60°, per cui conoscendo la posizione e l'orientamento (in riferimento alle informazioni acquisite dal robot) si può tracciare il cono di visione da esso proiettato.

Per semplicità si è scelto di creare una rappresentazione in 2 dimensioni, senza considerare la componente di altezza degli oggetti.
L'euristica che si è ideata si basa sull'assunzione di conoscere la dimensione minima e massima degli oggetti presenti nell'arena anche se ciò vale soltanto per gli oggetti sferici facenti parte della competizione. Infatti conoscendo lo spazio massimo e minimo che può occupare un oggetto nel campo visivo dell'agente, è possibile fare una stima di dove esso si trova. Per esempio, se l'agente ha nel suo campo visivo una sfera verde che appare piccola, potrebbe essere piccola e vicina oppure grande e lontana. Con opportuni calcoli si potrà quindi tracciare una zona in cui l'oggetto potrebbe trovarsi, poi man mano che l'agente esplora l'arena le stime fatte verranno migliorate grazie al maggior numero di informazioni raccolte.
L'idea di base è quindi la seguente:

- **Scegliere l'insieme di colori da considerare:** l'approccio si basa principalmente sul localizzare e memorizzare la posizione delle sfere di reward (positivo o negativo), le quali hanno colori ben precisi. Per cui quando si vedrà un insieme di punti di colore verde si andrà a stimare la zona dell'arena dove potrebbe essere in base a quanto appare grande nell'immagine acquisita dall'agente in un certo istante temporale. Nella competizione quasi tutti gli oggetti hanno colori diversi ed è quindi accettabile fare questa approssimazione. E' pertanto ovvio che sopratutto nella categoria 6 (vedi sezione 3.3.4) questo metodo potrebbe incorrere in alcuni problemi.

- **Definire la mappa:** bisognerà predisporre una struttura dati che possa contenere le informazioni relative alle stime del contenuto di ogni cella. Questa mappa avrà quindi dimensioni 112x112 per le motivazioni spiegate nella sezione precedente e disporre per ogni elemento di un numero di canali pari al numero di colori scelto. L'idea è che ogni volta che si stima che in una cella possa essere contenuto un oggetto di un certo colore si aumenta il rispettivo contatore per quel colore di quella cella.

- **Tenere traccia del movimento e disegnare la mappa**: utilizzando il modulo descritto in precedenza si può tenere traccia della posizione dell'agente, in modo da disporre delle stime fatte ad ogni passo temporale nella giusta porzione di mappa.

4.4.2.1 *Procedimento*

Per stimare l'area dovrebbe potrebbe essere incluso un determinato oggetto bisogna innanzitutto individuare punti del colore ricercato (poniamo ad esempio il verde) contigui fra loro, possiamo pensare una linea di punti verdi vicini come parte di una sfera di reward. Per trovare questi punti prima si cercano i punti verdi nell'immagini catturata:

```
    def _get_colours(self, pixels, colour):
        return np.all(pixels > np.minimum(colour * .8,
    colour - 25), axis=-1) & np.all(
            pixels < np.maximum(colour * 1.2, colour + 25),
     axis=-1)

    allPoints = np.zeros(shape=(84,84,len(self.colors)))
    obs = visualObs*255
    for i in range(0,len(self.colors)):
        allPoints[:,:,i] = self._get_colours(obs,self.
    colors[i])

```

Ottenuti i punti si vanno a ricercare quelli contigui in una stessa riga, considerando la loro posizione d'inizio nella riga e la lunghezza di questi segmenti di punti contigui. L'obiettivo a questo punto è di calcolare gli estremi della sezione del cono di visione, ovvero un trapezio. L'area compresa in questo trapezio sarà l'area dove potrebbe trovarsi un certo oggetto. Per spiegare i vari passi verrà fatto riferimento anche alla figura 31.

Per farli si considerano i due casi estremi, ovvero la sfera più piccola

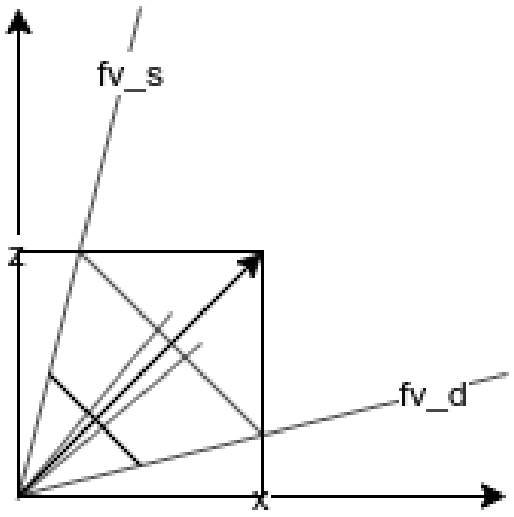


Figura 31: In questo piano cartesiano l'agente è in posizione (0,0) con un angolo di orientazione di 45˚, il suo campo visivo è compreso fra le linee fv_s e fv_d. Il quadrato rappresenta l'arena e la freccia che attraversa la sua diagonale è l'asse frontale del campo di visione dell'agente, le linee perpendicolari a quest'ultimo indicano le possibili ampiezze del campo visivo. Le linee più interne racchiudono invece l'area dove potrebbe trovarsi un ipotetico oggetto.

possibile, di diametro 0.5 e quella più grande, di diametro 5. Innanzitutto va calcolata quindi la distanza stimata fra il centro dell'oggetto della dimensione considerata e l'osservatore. Si denoti la distanza in questione con la lettera b, con a uno dei segmenti delimitanti il cono di visione(fv_d ad esempio) , c metà del segmento perpendicolare all'orientamento dell'agente che interseca il segmento b ed a. Infine

α è metà dell'angolo di visione e β l'altro angolo non retto. Quindi $\alpha = 30^{\cdot}, \beta = 60^{\cdot}$.
c è così ricavabile, (con value[1] lunghezza del segmento di punti contigui):

```
    original_size = value[1]/84
    for i in range(0,2):
        size = 0.5+(i*4.5)
        estimated_baseline=np.clip(size/original_size,0,43)

```

Dove estimated_baseline è $2c$.

$$b = a \sin \beta$$
$$c = a \sin \alpha$$
$$a = \frac{c}{\sin \alpha} = \frac{c}{2}$$
$$b = \frac{c}{2} \sin 60^{\cdot}$$

Quindi sapendo che estimated_baseline equivale a $2c$, la distanza $b = \textit{estimated\_baseline} \times \sin 60^{\cdot}$ A questo punto è possibile calcolare la lunghezza di uno dei segmenti delimitanti il cono di visione(fv_d ad esempio) tramite il teorema di Pitagora, avendo a disposizione gli altri due lati del triangolo:

```
        fv_line= np.sqrt(np.sum(np.square([distance,
    estimated_baseline/2])))

```

Quindi per sapere le coordinate dei punti di incrocio fra i lati del cono di visione e la retta perpendicolare all'orientamento dell'agente a distanza estimated_baseline basta usare le classiche formule trigonometriche:

```
    fv_l_x = np.cos(np.deg2rad(angle+30))*fv_line + my_posx
    fv_l_y = np.sin(np.deg2rad(angle+30))*fv_line +
    my_pos_y

    fv_r_x = np.cos(np.deg2rad(angle-30))*fv_line + my_posx
    fv_r_y = np.sin(np.deg2rad(angle-30))*fv_line +
    my_pos_y

```

Si hanno a questo punto gli estremi entro i quali è incluso il cono di visione, a questo punto non resta che calcolare gli estremi dell'oggetto secondo la stima minima e massima.
Di seguito il codice sviluppato per eseguire questo calcolo:

```
if fv_l_x <= fv_r_x:
    min_x = fv_l_x
    max_x = fv_r_x
    else:
        min_x = fv_r_x
        max_x = fv_l_x
if fv_l_y <= fv_r_y:
    min_y = fv_l_y
    max_y = fv_r_y
```

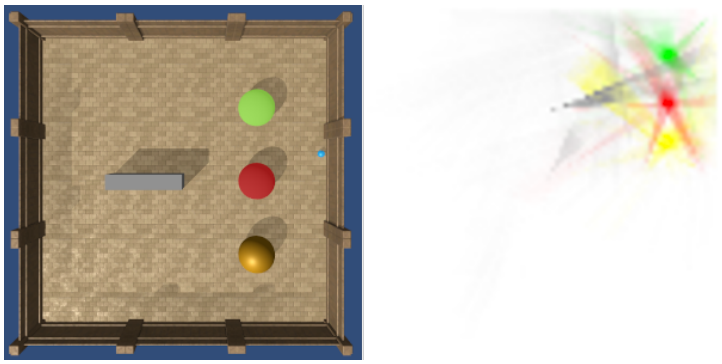
```
    else:
        min_y = fv_r_y
        max_y= fv_l_y
baseline_angle = np.absolute(90 - angle)
if baseline_angle >90:
    baseline_angle = baseline_angle %90

color_starts_at = (index/84)*estimated_baseline
value_size = (value[1] / 84) * estimated_baseline

if (0 <= angle <= 90) or (270 <= angle < 360):
    hypotenuse_left = estimated_baseline - color_starts_at
    hypotenuse_right = estimated_baseline - (
    color_starts_at+value_size)

if (90 < angle <= 180) or (180 < angle < 270):
    hypotenuse_left = color_starts_at
    hypotenuse_right = color_starts_at + value_size

left_x = hypotenuse_left * np.cos(np.deg2rad(baseline_angle
    ))
left_y = hypotenuse_left * np.sin(np.deg2rad(baseline_angle
    ))

right_x = hypotenuse_right * np.cos(np.deg2rad(
    baseline_angle))
right_y = hypotenuse_right * np.sin(np.deg2rad(
    baseline_angle))

if (0 < angle <= 90) or (180 < angle < 270):
    left_point_x = max_x -left_x
    left_point_y = min_y+left_y
    right_point_x = max_x -right_x
    right_point_y = min_y+ right_y

if (90 < angle < 180) or (270 <= angle < 360):
    left_point_x = min_x+left_x
    left_point_y = min_y+left_y
    right_point_x = min_x +right_x
    right_point_y = min_y + right_y

if angle == 0 or angle == 180:
    left_point_x = min_x
    left_point_y = min_y + left_y
    right_point_x = min_x
    right_point_y = min_y + right_y

x_1 = np.clip(left_point_x+half_map,0,mapSize-1)
x_2 = np.clip(right_point_x+half_map,0,mapSize-1)
y_1 = np.clip(left_point_y+half_map,0,mapSize-1)
y_2 = np.clip(right_point_y+half_map,0,mapSize-1)

```

In pratica per calcolare gli estremi dell'oggetto secondo la stima minima e massima bisogna quindi considerare anche l'orientazione dell'agente ed a seconda di essa si fanno gli opportuni calcoli per ottenere le coordinate del poligono da tracciare.

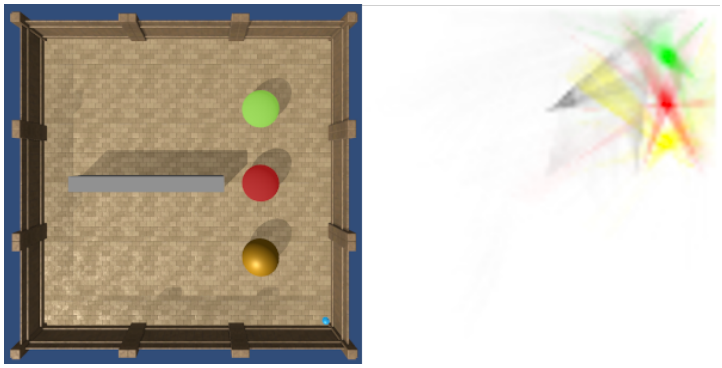
4.4.3 *Risultati e limiti dell'approccio*

Verranno ora riportati e discussi alcuni risultati di mappatura dell'arena secondo l'approccio descritto. Va ricordato che il metodo potrà essere abbastanza affidabile per quanto riguarda le sfere, le quali hanno dimensioni in un certo intervallo, ma necessariamente per tutti gli altri oggetti la stima non può essere precisa, in quanto, ad esempio, un muro può essere lungo anche quanto tutta l'arena. Nella figura 32 sono riportati alcuni esempi di mappe costruite con questo approccio, facendo costeggiare tutto il bordo dell'arena all'agente e facendo ruotare su stesso in modo da poter vedere l'arena. Per costruire le mappe queste mappe l'agente ha eseguito 550 step di movimento.

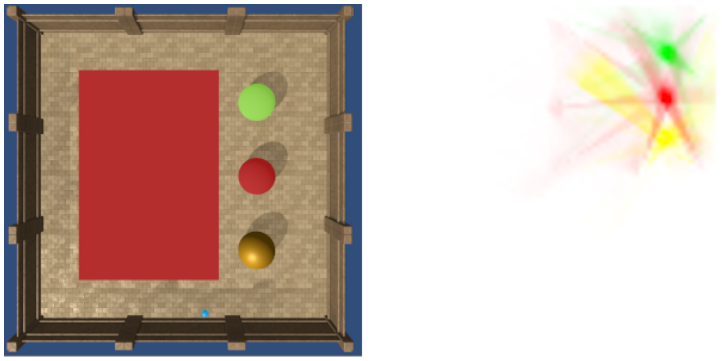
Si può notare che la mappa è abbastanza accurata nel caso in cui contenga solo le sfere di reward, ma meno accurata se vi sono altri elementi che non hanno un range di dimensioni fissato, proprio per le considerazioni già fatte in precedenza.
Inoltre è bene evidenziare che le mappe realizzate saranno accurate solo se l'agente riesce ad avere informazioni da più punti di vista, triangolando di fatto le stime. Anche nei casi in cui il mapping è più preciso, come nella figura 32-d nelle stime è presente molto rumore, il che potrebbe influire sull'utilità in fase di training. Altro problema, collegato al precedente, è che serviranno molti step per creare una rappresentazione sufficientemente fedele, ma come sappiamo in generale la durata degli episodi è limitata ed abbastanza breve (250 passi).
Sono stati perciò condotte delle indagini preliminari per capire se l'algoritmo di reinforcement learning utilizzato (di cui si parlerà nel prossimo capitolo) potesse riuscire ad apprendere una buona policy a partire da queste mappe create euristicamente, anche al fine di orientarsi verso un approccio di RL Model-Based. Le indagine condotte hanno indicato che i modelli risultanti non fossero del tutto maturi per essere utilizzati nella competizione vera e propria e considerate le stringenti scadenza imposta dalle varie fasi della competizione si è deciso di orientarsi verso approcci di più immediato risultato.



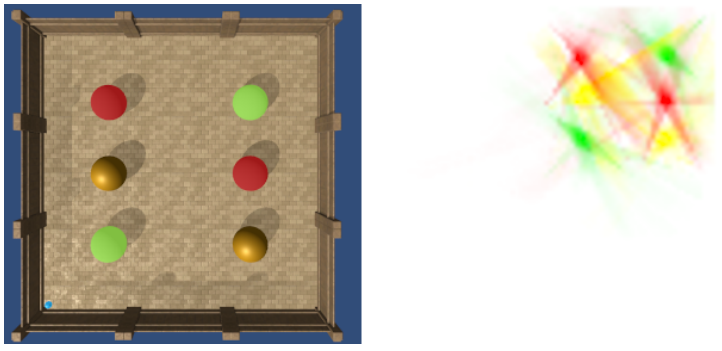
(a) *Mappa di un arena con tre sfere di colore diverso ed un muro lungo 10 metri*



(b) *Mappa di un arena con tre sfere di colore diverso ed un muro lungo 20 metri*



(c) *Mappa di un arena con tre sfere di colore diverso ed una zona mortale di 20x30 metri*



(d) *Mappa di un arena con tre coppie sfere di colore diverso*

Figura 32: Alcuni risultati del metodo euristico per il mapping dell'arena

5

SOLUZIONE PROPOSTA

Questo capitolo rappresenta il cuore di tutto il lavoro svolto, in quanto verrà descritto in dettaglio come si è deciso di affrontare le sfide poste dalla competizione Animal-AI Olympics. Come da buone pratiche si partirà trattando gli aspetti metodologici, stilando una lista dei requisiti che un buon agente software dovrebbe soddisfare per ottenere un buon risultato nella competizione.
Una volta raccolti i possibili requisiti si procederà nell'analizzarli, evidenziando quali siano più soddisfacibili e quali invece dovranno essere affrontati in eventuali lavori futuri. Verranno fatti alcuni cenni sugli strumenti tecnologici messi in campo, in modo da fornire una visione completa del processo di sviluppo al lettore.
Dopo aver introdotto cosa sarà necessario fare, verranno descritte le componenti software realizzate, partendo da una parte fondamentale quale l'infrastruttura di test locale per proseguire verso la descrizione delle strategie messi in campo per addestrare l'agente in modo più efficace e flessibile possibile.
L'ultima sezione del capitolo tratterà in dettaglio gli aspetti legati all'applicazione del reinforcement learning al dominio della competizione, evidenziano le strategie messe in campo, discutendone utilità ed efficacia.

5.1 ASPETTI METODOLOGICI

Come da prassi nel campo dell'ingegneria, è bene che prima di realizzare una soluzione si stilino ed analizzino quali sono i requisiti che questa soluzione dovrà soddisfare. Alcuni requisiti saranno a livello di infrastruttura di addestramento e test, altri saranno legati alla performance dell'agente, altri ancora sulle capacità cognitive che esso dovrebbe possedere. Ovviamente alcuni di questi requisiti non possono che rimanere dei desiderata, vista la complessità del dominio applicativo, ma in ogni caso per ognuno di essi verrà fatta una oculata riflessione.

5.1.1 *Requisiti*

Prima ancora di elencare i requisiti che l'agente software da utilizzare nella competizione dovrà esibire, bisognerà specificare quali sono le

proprietà che dovranno avere le infrastrutture di training e test. E' infatti fondamentale che prima di partire con i vari esperimenti si abbia a disposizione una seria infrastruttura di test locale, attraverso la quale valutare la bontà dell'agente addestrato prima di inviarlo per la valutazione nella competizione, si ricordi infatti che il numero di submission giornaliere autorizzate è di 3, perciò non si potrà fare affidamento solo alla valutazione ufficiale per valutare la bontà degli approcci.
Questa infrastruttura di test dovrà perciò esibire i seguenti requisiti:

1. Utilizzare lo stesso template di agente che si utilizzerà per essere valutati nella competizione.

2. Dovrà essere possibile aggiungere ed eliminare configurazioni di test facilmente.

3. Dovrà essere fornita la possibilità di stabilire una soglia di ricompensa per cui l'episodio è considerato superato, così come avviene nella competizione ufficiale.

4. Dovrà essere possibile specificare con facilità su quali categorie eseguire il test e quanti episodi includa ognuna di esse.

5. Dovrà essere possibile testare modelli di agente anche molto diversi tra loro in maniera totalmente trasparente, senza dover cambiare l'Infrastruttura complessiva.

6. Al termine di un test dovrà essere riportato il risultato ottenuto, sia in forma visiva, sia, sopratutto su un apposito file. Questo file dovrà contenere sia i punteggi complessivi per ogni categoria sia il report completo su quali episodi sono stati superati.

7. Dovranno essere sviluppati un numero di test significativo, cercando di capire al meglio quali potrebbero essere le situazioni da testare.

8. I test locali dovrebbero essere ideati in modo che il risultato ottenuto su di essi da un certo agente rispecchi, almeno sommariamente, quello ottenuto dallo stesso agente nei test ufficiali.

Dopo la struttura di test è sicuramente fondamentale identificare quali siano le caratteristiche desiderabili da un architettura software da mettere in campo per addestrare degli agenti di RL:

1. A prescindere dalla tipologia di algoritmo di RL che si utilizzerà esso dovrà essere facilmente configurabile ed adattabile al dominio applicativo.

2. Dovrà essere possibile lanciare un esperimento di addestramento specificando tutti i possibili iper-parametri e configurazioni utilizzabili tramite riga di comando o altra soluzione che permetta di non cambiare il codice sorgente ad ogni nuovo addestramento.

3. Durante l'addestramento dovrà essere possibile salvare lo stato del modello con una frequenza specificata nella configurazione dell'esperimento.

4. Per addestrare l'agente dovranno essere messe a disposizione delle specifiche configurazioni dell'arena, le quali dovranno essere divise in categorie e ideate in modo da favorire un addestramento robusto ed efficace.

5. Sarebbe utile disporre di una fase di validazione, eseguita ciclicamente ogni tot epoche di addestramento, al fine di valutare l'andamento del processo di addestramento. Questi risultati intermedi dovrebbero essere memorizzati in appositi file, in modo da poterli consultare ed analizzare in un secondo momento.

6. Le possibile strategie utilizzate per addestrare ed implementare l'agente non devono influire sulla struttura dell'architettura di base.

7. L'architettura di addestramento risultante dovrebbe essere realizzata in modo da poter essere eseguita su qualsiasi macchina cambiando solo qualche configurazione da riga di comando.

Quanto appena scritto si riferisce principalmente ad aspetti architetturali, legati all'ovvia esigenza di realizzare un sistema che permetta di addestrare e testare vari approcci senza perder tempo in problematiche non legate al domino del RL.
Verranno ora analizzati i requisiti, sopratutto cognitivi che dovrebbe soddisfare l'agente per superare gran parte dei test di ogni categoria. Considerato che le categoria non sono tutte della stessa difficoltà, è bene separare queste caratteristiche desiderabili in base alla categoria in cui sono richieste (per dettagli sulle categorie di test fare riferimento alla sezione 3.3.4:

1. L'agente dovrà avere la capacità di discernere fra sfere che attribuiscono ricompense e quelle che invece infliggono penalità, riuscendo a raggiungere le prime ed evitando le seconde, muovendosi con efficacia in arene vuote.

2. L'agente dovrà dimostrare delle preferenze verso situazioni più premianti, anche se più lontane nel tempo. Per farlo dovrà apprendere la differenza fra le varie tipologie di sfere.

3. L'agente dovrà intuire gli effetti dei vari possibili oggetti introdotti in questa categoria sul suo movimento. Un muro, ad esempio, non può essere attraversato e se non trasparente non permette di vedere cosa c'è dietro; una rampa può permettere all'agente di salire ad una altitudine diversa da quella iniziale. Alcuni oggetti non potranno essere spostati dall'agente, mentre altri si, e sarebbe bene che esso capisca quali di essi sono spostabili, in modo da non

perdere tempo a spostare, ad esempio, un tunnel. In definitiva l'agente dovrà essere capace di navigare in un arena con diversi possibili oggetti, cercando di ottenere il cibo che lo premia.

4. In questo caso l'agente dovrebbe avere già il concetto di equivalenza fra il colore rosso e la terminazione immediata dell'episodio con annessa penalità come nella prima categoria. In questo caso però si tratta di zone poste sul terreno, perciò meno visibili sopratutto da lontano e potenzialmente molto più estese delle semplici sfere. L'agente dovrà inoltre imparare l'associazione fra le zone arancione ed una penalità pari al tempo trascorso su di esse.

5. Per superare questa tipologia di test l'agente dovrà disporre di una forte comprensione della fisica con cui funziona l'ambiente e sopratutto avere un qualche tipo di memoria spaziale che lo aiuti nella navigazione. In questo caso una rappresentazione mentale dell'arena potrebbe essere fondamentale.

6. Per potere riuscire a superare i test di questa categoria in cui non vengono rispettate le convenzioni cromatiche che valgono per il resto delle categorie, l'agente dovrebbe avere una capacità di riconoscere i vari oggetti presenti a prescindere dal loro colore.

7. Per raggiungere il proprio obiettivo pur non avendo a disposizione la vista per un certo lasso temporale, l'agente deve necessariamente memorizzare lo stato dell'ambiente magari riuscendo a prevedere gli stati futuri dello stesso.

8. In questa categoria di test è fondamentale disporre del concetto di permanenza dell'oggetto, ovvero la capacità di ricordarsi dell'esistenza di un certo elemento anche se non è più visibile. Ciò quindi richiede che l'agente abbia una rappresentazione interna strutturata degli oggetti.

9. Di fatto per avere successo in questa categoria l'agente deve aver la capacità di valutare le possibili opzioni da intraprendere per trovare il percorso migliore ed ottenere più punteggio.

10. Trattandosi della categoria più complessa di tutta la competizione l'agente dovrà disporre di un solido nucleo di ragionamento casuale, che gli permetta di valutare le possibile azioni da mettere in campo e gli effetti delle stesse, in modo da raggiungere l'obiettivo in situazioni in cui un alto grado di capacità cognitiva è richiesta.

5.1.2 *Analisi dei requisiti*

Dopo aver stilato tutti i requisiti che dovrebbero esibire l'agente intelligente e l'architettura software a supporto del suo addestramento e testing, si procederà ora nell'analizzare i vari requisiti, valutandone

criticità e possibili strategie.

5.1.2.1 *Infrastruttura di test*

Per quanto riguarda l'infrastruttura di test essa dovrà essere implementata prima di ogni altra cosa, in quanto senza di essa non si potrà valutare la bontà dell'approccio scelto e perciò sarebbe difficile scegliere quali agente inviare per la valutazione.
Per quanto riguarda i relativi requisiti espressi nella sezione precedente di seguito sono riportate le considerazioni fatte:

1. Gli organizzatori della competizione hanno messo a disposizione un template di agente da seguire obbligatoriamente per poter essere valutati. Esso è molto semplice ed intuitivo e può essere utilizzato senza problemi anche in questa fase di test locale.

2. Dato che le configurazioni dell'arena devono essere necessariamente specificate tramite file YAML attraverso un apposita sintassi, questo può essere sfruttato anche nella fase di test, andando a definire tanti file di configurazione, divisi per categoria. Si può pensare ad ogni categoria sia legata una cartella, dove prendere le configurazioni da utilizzare.

3. Per considerare un test superato bisognerebbe stabilire una soglia di punteggio da raggiungere, ma ovviamente ad ogni file di configurazione dell'arena dovrà essere legata una soglia specifica, definita dal progettista. Si può pensare già in fase di analisi che potrebbe essere vantaggioso includere questa informazione del file yaml, in modo da mantenere tutto insieme e facilitarne la successiva modifica.

4. Considerato che a volte potrebbe essere utile testare l'agente solo per certe categorie, sarà bene rendere facilmente specificabile questa scelta, il che non dovrebbe comportare particolari problematiche.

5. Per utilizzare agenti implementati secondo approcci abbastanza diversi bisognerà mettere in campo una serie di meccanismi che permettano con facilità di cambiare da una tipologia di agente ad un altro, verrà ovviamente comodo utilizzare lo scheletro di agente introdotto al punto 1.

6. Per riportare il risultato su file si dovrà fare ben attenzione a rendere il più semplice possibile un analisi successiva. Alcuni accorgimenti utili potrebbero includere la denominazione del file secondo un nome significativo, magari definito nella configurazione dell'addestramento, specificando anche il numero di iterazioni di addestramento dopo cui il modello è stato salvato su file.

7. Nella competizione, almeno nella fase principale, esistono 30 test per ogni categoria, il che ha richiesto sicuramente un grande lavoro agli organizzatori, in quanto non è affatto banale ideare dei test che dimostrino una certa capacità cognitiva dell'agente. Dato che sviluppare 300 test, di cui alcuni la progettazione potrebbe richiedere molto tempo (quelli più complessi), porterebbe via troppo tempo utile, si è deciso intanto di implementare 30 test per ognuna delle prime 4 categorie, in modo da avere un benchmark almeno sulle categorie più semplici, in quanto se non si riescono a superare queste sarà difficile poter superare quelle successive.

8. Garantire questo requisito è molto complicato perché i test sono nascosti, per cui solo confrontando i risultati dei test condotti in locale e quelli ottenuti poi nella valutazione ufficiale, si potrà avere una stima di quanto fedeli e significativi siano i test sviluppati in locale.

5.1.2.2 *Infrastruttura di addestramento*

Per quanto riguarda l'infrastruttura di addestramento essa rappresenta sicuramente il secondo fondamentale della soluzione complessiva da mettere in campo, essa infatti sarà il cuore del progetto, dando la possibilità di addestrare un agente di RL seguendo vari approcci.

- Dato che implementare da zero un algoritmo di RL allo stato dell'arte richiederebbe un tempo davvero consistente, non compatibile con le tempistiche dettate dalla competizione, sarebbe preferibile utilizzare qualche implementazione open-source di baseline, che magari si adatti con degli ambienti che seguano l'interfaccia di OpenAI Gym (vedi sezione 3.2.2). Una buona scelta potrebbe essere rappresentata dalle cosiddette ***Stable baseline***[12], un fork di OpenAI baseline che contraddistingue per chiarezza e robustezza delle implementazioni, il tutto corredato da una documentazione efficace.

- Tramite le apposite utility di Python non dovrebbe essere un problema specificare tutti gli iperparametri ed i possibili aspetti configurabili dell'addestramento. Si può già pensare in fase di analisi che sarà preferibile disporre di appositi script dove specificare il tutto e lanciare gli esperimenti, in modo da tenere traccia della configurazione scelta.

- Riguardo al salvataggio del modello addestrato qualsiasi framework di machine learning e qualsiasi baseline offre la possibilità di salvare lo stato del modello in qualsiasi momento.

- Per l'addestramento dell'agente dovranno essere messe in campo configurazioni concepite diversamente dalle configurazioni di test. Esse dovranno infatti essere il più generali possibili ed includere

molti elementi casuali, in modo da aumentare la generalizzazione dell'agente. Sarà però importante definire questi scenari di training oculatamente, in modo da favorire un addestramento robusto. Fortunatamente l'ambiente della competizione è realizzato in modo che si possa lasciare al caso la scelta di alcuni parametri della configurazione.

- In questo caso si dovrà includere una fase di validazione ad intervalli stabiliti, nella quale l'agente sfrutti il modello appena addestrato per svolgere un insieme specificato a priori di test, riportando i risultati relativi ad una certa iterazione, sia su di un file apposito sia su di uno complessivo da cui si possa desumere l'andamento dell'addestramento nel tempo.
- Per rendere l'architettura di addestramento il più slegata possibile dal approccio di RL scelto, sarà bene effettuare una efficace separazione dei componenti della soluzione, cercando di mantenere intatta la struttura originale e rendendo possibile cambiare approccio tramite il file di configurazione specificato.
- Per rendere facilmente portabile l'architettura di addestramento potrebbe essere messo in campo uno strumento molto utile quale Docker [21].

5.1.2.3 *Capacità cognitive dell'agente*

Per quanto riguarda i requisiti legati alle capacità cognitive dell'agente è ovvio che siano tutte desiderabili ed importanti, ma il raggiungimento di alcune di queste proprietà è l'obiettivo del campo dell'intelligenza artificiale da svariate decadi e sono realisticamente difficili da raggiungere. Si procederà quindi per gradi, cercando di soddisfare innanzitutto i requisiti posti dalle categorie 1,2 e 4, le quali includono principalmente la capacità di distinguere stimoli positivi e negativi e riuscire a soddisfarli o, nel secondo caso, evitarli. Altre categorie verranno comunque "attaccate", ma ovviamente considerando le difficoltà intrinseche in esse.

5.1.3 *Strumenti tecnologici*

Nella soluzione proposta di è deciso di utilizzare il linguaggio di programmazione Python, che oltre ad essere il linguaggio richiesto per l'invio dei vari agenti da testare nella competizione, ci offre la possibilità di usufruire di svariate librerie, nello specifico, tra quelle utilizzate sono degne di nota:

- **TensorFlow**: è una libreria software open source per il calcolo numerico che utilizza grafici di flusso di dati. I nodi del grafico rappresentano operazioni matematiche mentre gli arche del grafo rappresentano gli array di dati multidimensionali (ovvero i tensori) comunicati tra di essi. L'architettura flessibile consente di

distribuire il calcolo ad una o più CPU o GPU in un desktop, un server o un dispositivo mobile con una singola API.

- **Numpy** è una libreria che aggiunge il supporto per matrici e grandi array multidimensionali, insieme ad una vasta collezione di funzioni matematiche di alto livello per operare su questi array.

- **OpenAI Gym:**, descritto nella sezione 3.2.2, questo toolkit permette di facilitare lo sviluppo ed il benchmark di algoritmi di RL.

- **OpenCV:** è una libreria software per la visione artificiale ed il machine learning, essa include più di 2500 algoritmi ottimizzati per i più svariati task della materia.

E' stato inoltre utilizzato Docker, sia per l'invio dell'agente per la competizione (in quanto obbligatorio), sia per addestrare l'agente su macchine diverse. Grazie ad esso è infatti possibile configurare un contenitore virtuale utilizzabile su diverse macchine senza dover cambiare di una virgola il codice sorgente e semplificando enormemente il deploy, che in questi casi richiede molto tempo essendo necessaria l'installazione di vari software e librerie specifiche.
Per l'addestramento sono state utilizzate due diverse macchine:

- Un server, il Raptor02, gentilmente messo a disposizione dal BioLab di Cesena, con a bordo due GPU: una Titan X Pascal ed una GTX Titan X, entrambe con 12GB di memoria, oltre ad un processore con 12 core e 64 Gb di RAM.

- Per una decina di giorni si è utilizzata un istanza AWS del tipo g3.8xlarge, dotata di due GPU Tesla M60, 244 Gb di Ram e 32 CPU.

5.2 COMPONENTI DEL PROGETTO

In questa sezione verranno illustrati i vari componenti implementati secondo i requisiti e le considerazione precedentemente fatte.

5.2.1 *Infrastruttura per test locale*

5.2.1.1 *Configurazione dei test*

Per sviluppare la piattaforma per testare i vari modelli di agente addestrati è stato innanzitutto necessario trovare il modo di definire una soglia di superamento per ogni configurazione di test.
Si è ritenuto che la scelta migliore potesse essere quella di includere questa soglia direttamente nel file YAML di specifica dell'arena, a titolo esemplificativo di seguito viene riportato uno di questi file:

```
!ArenaConfig
arenas:
  0: !Arena
    t: 250
    items:
    - !Item
      name: GoodGoalMulti
      sizes:
        - !Vector3 {x: 5, y: 0, z: 30}
      positions:
        - !Vector3 { x: 35, y: 0, z: 35}
    - !Item
      name: GoodGoalMulti
      sizes:
        - !Vector3 {x: 4, y: 0, z: 25}
      positions:
        - !Vector3 { x: 20, y: 0, z: 20}
    - !Item
      name: GoodGoalMulti
      sizes:
        - !Vector3 {x: 3, y: 0, z: 35}
      positions:
        - !Vector3 { x: 3, y: 0, z: 35}
    - !Item
      name: Agent
      positions:
        - !Vector3 { x: 20, y: 0, z: 1}
      rotations: [0]
name: 'Collects all simple'
score_to_pass: 10.0
```

L'ultima riga del file è proprio quella che indica quant'è il punteggio che va superato per considerare il test come passato.
Per ottenere questa informazione dal file è stato necessario sviluppare un apposita funzione per leggere il valore del campo "score_to_pass", utilizzata nella funzione per caricare i test relativi ad una categoria specificata. Di seguito il codice delle due funzioni:

```
from yaml import load, dump
try:
    from yaml import CLoader as Loader, CDumper as Dumper
except ImportError:
    from yaml import Loader, Dumper

def getScoreThreshold(file_path):
    fileHandle = open(file_path,'r')
    lineList = fileHandle.readlines()
    index= -1
```

```
    lenFile = len(lineList)
    while not lineList[index].startswith("score_to_pass")
    and -index <lenFile:
        index += -1
    last_line = lineList[index]
    score = load(last_line,Loader)
    return score['score_to_pass']

def getArenaConfigsAndThresholdByCategory(index):
    index = str(index)
    base_path = "configs/c"+index
    onlyfiles = [join(base_path,f) for f in
    sorted_aphanumeric(listdir(base_path)) if isfile(join(
    base_path, f))]
    print(onlyfiles)
    configs = [(ArenaConfig(f), getScoreThreshold(f)) for f
     in onlyfiles]
    return configs

```

In definitiva l'ultima funzione presentata restituisce una lista di coppie, di cui il primo valore è un oggetto del tipo ArenaConfig (da passare all'ambiente della competizione per specificare il contenuto dell'arena) ed il secondo è proprio la soglia di punteggio da superare. Tutti i file di configurazione sono contenuti in un apposita cartella denominata *configs* e per ogni categoria corrisponde una cartella (ad esempio categoria 1 cartella *configs/c1/*).

5.2.1.2 *Configurazione di un test locale*

Lo scenario d'uso di questa infrastruttura prevede che un certo modello addestrato venga sottoposto ad un certo insieme di configurazioni di test appartenenti alle categorie che si è scelto di testare.
La soluzione implementata permette infatti di specificare quali categorie debbano essere testate e quanti test effettuare per ognuna di queste categorie.
Tra le opzioni specificate nell'avviare un test è possibile specificare se registrare le immagini provenienti dalle osservazioni dell'agente, specificando anche quali episodi registrare visto che non tutti potrebbero essere di interesse per gli scopi dell'utilizzatore.
Altra possibilità è quella di specificare un set di modelli da testare sequenzialmente secondo le medesime condizioni, il che risulta molto comodo se si hanno diversi diversi modelli che si vogliono valutare.
Esistono anche altre opzioni configurabili, ma esse riguardano tutte quali strategie debba utilizzare l'agente, come ad esempio specificare se l'agente usa una LSTM oppure no. Questa configurabilità permette di non dover cambiare l'infrastruttura principale di test.

5.2.1.3 *Loop di test*

Come si evincerà dalla porzione di codice riportato, il processo di valutazione locale dell'agente è organizzato in due cicli, quello che itera

le categorie e quello che itera i test relativi ad ogni categoria. Ogni episodio a sua volta può terminare o quando il numero di passi massimo è stato raggiunto o quando viene terminato anzitempo da una zona rossa o una sfera verde/rossa. Ad ogni episodio il punteggio si azzera e viene incrementato con i reward (positivi o negativi) ottenuti ad ogni suo passo, al suo termine questo valore viene confrontato con la soglia di superamento per determinare se l'episodio è stato svolto con successo. Le informazioni relative ai test superati vengono mantenute sia cumulativamente rispetto alle varie categorie, sia a livello di ogni singolo test affrontato. Nel codice è anche visibile la porzione che è adibita al salvataggio delle immagini acquisite dall'utente.

```
    print("..............................Starting test
    ..............................")
    succesful_episodes = 0
    for c in testedCategory:
        category_configs = getArenaConfigsAndThresholdByCategory(c + 1)
        # print(category_configs)
        print(f"----------------------Starting tasks of category {c
    +1}----------------------")
        for t in range(num_task):
            selected_config = category_configs[t][0]
            # print(selected_config,t)
            obs = env.reset(selected_config)
            submitted_agent.reset()
            tracker.resetTracker()
            cumulated_reward = 0
            try:
                obs, reward, done, info = env.step([0, 0])
                for i in range(selected_config.arenas[0].t):
                    action = submitted_agent.step(obs, reward, done, info)
                    obs, reward, done, info = env.step(action)
                    brain = info['brain_info']
                    velocities = brain.vector_observations
                    if save_images and episodes_to_save is not None:
                        if episodes_to_save[c][t] == 1:
                            if i == 0:
                                save_path = os.path.join(results_dir, "c"+
    str(c+1)+"-t"+str(t+1))
                                try:
                                    os.mkdir(os.path.join(save_path))
                                except OSError as error:
                                    print("Already existent directory")
                            visual_obs = brain.visual_observations[0][0]
                            image_path = os.path.join(save_path,"image" +
    str(i) + ".png")
                            plt.imsave(fname=image_path, arr=visual_obs)
                    cumulated_reward += reward
                    if done:
                        break
            except Exception as e:
                # print('Episode {} failed'.format(k))
                raise e
            if (cumulated_reward > category_configs[t][1]):
                scores[c]+=1
                complete_scores[c][t]+=1
                print(f"Task {t+1} of category {c+1} passed with a total
    reward of {cumulated_reward} points in {i} step")
            else:
                print(f"Task {t + 1} of category {c + 1} failed with a total
     reward of {cumulated_reward} points in {i} step")
            if show_plot:
                tracker.plotRoute()
            print(f"Passed {scores[c]} of {t+1} task")
        print(f"Category {c+1} finished with {scores[c]} successfull
    episodes")

    print("Test finished with score:",scores)
    write_result_file(scores,complete_scores,base_path,modelName)
```

Quando l'agente termina di essere valutato su tutti i test su cui andava testato viene scritto un apposito file contenente il risultato della valutazione. Il file è un json, ed un esempio di una porzione di esso viene riportato di seguito:

```
{
```

```
  "Result for categories": {
    "1": 25,
    "2": 24,
    "3": 0,
    "4": 17,
    "5": 0,
    "6": 0,
    "7": 0,
    "8": 0,
    "9": 0,
    "10": 0
  },
  "Complete results": {
    "1": {
      "1": 0,
      "2": 1,
      "3": 0,
      "4": 1,
      "5": 1,
      "6": 1,
      "7": 1,
      ...
```

Nel campo "Complete results" viene riportato il dettaglio episodio per episodio, riportando il valore 1 se esso è stato superato o 0 altrimenti.

5.2.1.4 *Lo scheletro dell'agente*

A prescindere dall'approccio utilizzato l'agente da testare ha un preciso scheletro da rispettare, nel quale sono definiti principalmente tre metodi: *step(obs, reward, done, info), reset(t), init()*.
Al fine di utilizzare in modo trasparente diverse tipologie di agente è stata realizzata una utility tramite la quale si può istanziare l'agente desiderato tramite appositi metodi. Ai fini della spiegazione non è però interessante riportare a questo punto il codice della utility in quanto legata ad aspetti implementativi degli approcci adottati.

5.2.2 *Configurazioni ambienti per test*

Come stabilito in fase di analisi dei requisiti nella sezione 5.1.2.1 punto 7 e 8, si sono ideati 30 test per ognuna delle prime 4 categorie della competizione. Come già introdotto lo sviluppo di un test che dimostri il possedimento di determinate capacità cognitive non è banale, richiede infatti che i test siano fatti in modo che comportamenti casuali dell'agente non permettano di raggiungere l'obbiettivo, bensì deve esse chiaro vedendo l'episodio che l'agente ha una certa capacità cognitiva.
Per sviluppare i test si sono seguite alcune delle indicazioni fornite dagli organizzatori della competizione, ma al contempo cercando di minimizzare la differenza di risultato fra i test locali e quelli ufficiali. Durante tutta la competizione si è infatti tenuto traccia dei risultati sia ufficiali che locali di ogni modello inviato per valutazione, facendo

ogni volta un confronto per capire quali dei test sviluppati fossero più significativi.
I 120 test sviluppati inizialmente sono stati quindi raffinati e cambiati da restituire risultati il più significativi possibile, cosicché un aumento nel punteggio sui test locali corrispondesse poi ad un aumento del punteggio ottenuto nei test ufficiali della competizione.
Verranno ora mostrati alcuni esempi delle configurazioni di test, illustrando i principi che hanno guidato l'ideazione dei test per le prime quattro categorie.

5.2.2.1 *Categoria 1*

Per quanto riguarda la categoria 1 i test sviluppati sono abbastanza semplici, come prescritto anche dagli organizzatori della competizione. Nell'idearli si è voluto verificare che l'agente sia in grado di:

- andare verso una sfera verde di varie dimensioni posta frontalmente ad esso a varie distanze. Nei task più semplice la sfera è abbastanza grande, in quelli più complessi la sfera costituisce solo pochi pixel dell'immagine percepita dall'agente.

- raccogliere tutte le sfere gialle presente in un arena, in modo da massimizzare il punteggio e terminare l'episodio.

- evitare le sfere rosse in assoluto, ma non fermandosi davanti ad esse, bensì dirigersi verso altre aree e prioritariamente verso sfere verdi/gialle. In alcuni casi le sfere rosse sono nella direzione di quelle verdi, in altri sono dietro di esse, questo perché si vuole testare che l'agente sappia capire in quali situazioni è sicuro andare a recuperare una sfera di reward positivo. Si vuole infatti evitare che appena l'agente veda il rosso rimanga fermo per sempre.

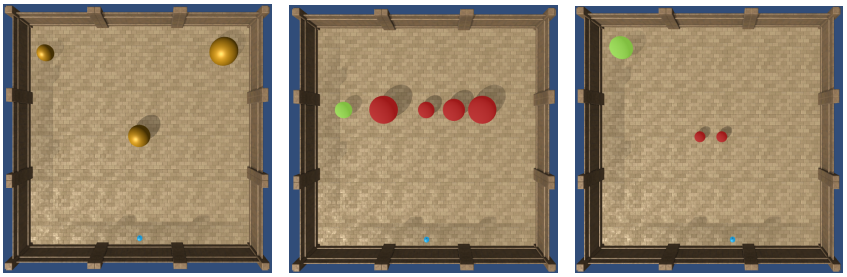
- di raccogliere sfere in movimento.

- di trovare una sfera di reward in un arena vuota, anche se non direttamente visibile dallo stato iniziale dell'agente.

5.2.2.2 *Categoria 2*

In questo caso i test da sviluppare richiedono maggior ragionamento, in quanto bisogna concepire dei test in cui una scelta sia oggettivamente preferibile ad un'altra. Altra difficoltà sta nel quantificare una soglia di punteggio per cui si può dire che l'episodio è stato superato. Nell'ideare questi test si è voluto verificare che l'agente sia in grado di:

- scegliere sempre una sfera verde più grande rispetto ad una più piccola, sopratutto se sono vicine tra loro.



(a) *Test 2 - Raccogliere tutte le sfere gialle*

(b) *Test 19 - Raccogliere la sfera verde evitando le rosse*

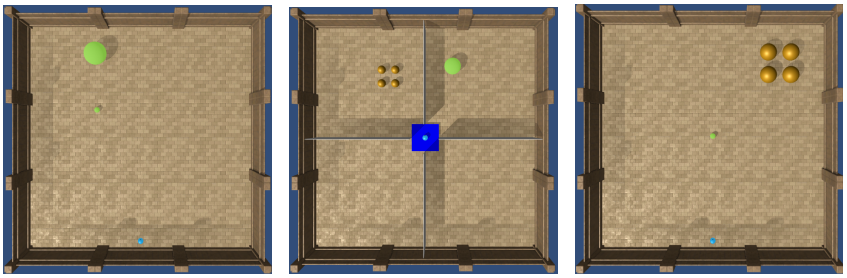
(c) *Test 30 - Raccogliere la sfera verde all'angolo evitando le rosse in centro*

Figura 33: Tre esempi di test della categoria 1. Nel test 2 l'episodio è considerato superato solo se si raccolgono tutte e tre le sfere, mentre negli altri due è sufficiente raccogliere la sfera prima dello scadere del tempo, ovvero prima di 250 step.

- scegliere, se ad esempio sono presenti una sfera gialla ed una verde, sempre prima la gialla e poi la verde, portando a massimizzare il punteggio, in quanto prendendo prima la sfera verde non sarebbe possibile prendere la gialla perché l'episodio termina. Questo concetto viene ovviamente declinato in varie forme, ma l'idea di base è quella di scegliere una sequenza di azioni che permetta di raccogliere più ricompense.

- scegliere, se nell'arena vi sono una sfera piccola e vicina ed una grande ma più lontana, sempre quella più grande, in modo da massimizzare il punteggio.

- scegliere il percorso più veloce per arrivare ad una sfera



(a) *Test 4 - Raccogliere la sfera più grande*

(b) *Test 13 - Raccogliere 4 sfere piuttosto che una*

(c) *Test 19 - Ottenere molta più ricompensa anche se più lontana*

Figura 34: Tre esempi di test della categoria 2. Nel test 4 l'episodio è considerato superato solo se si raccoglie la sfera più grande, mentre negli altri due l'agente deve raccogliere tutte e 4 le sfere gialle per superare il test.

5.2.2.3 *Categoria 3*

In questa categoria della competizione vengono introdotti tanti nuovi oggetti nell'arena, richiedendo perciò di testare tante varie situazioni. Nei vari test sono stati infatti utilizzati tunnel e muri (trasparenti e non), scatole varie e rampe per salire su piattaforme. Sebbene teoricamente questa rientri nelle categorie più semplici in realtà superare tutte le sfide poste dall'utilizzo dei vari nuovi oggetti non è affatto banale. Nell'ideare questi test si è voluto verificare che l'agente sia in grado di:

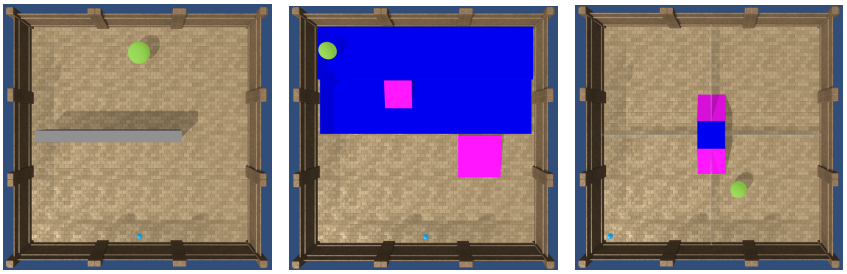
- trovare sfere gialle/verdi poste dietro mura non trasparenti o dentro tunnel.
- utilizzare le rampe per salire su piattaforme sulle quali potrebbero essere presente delle ricompense.
- spostare oggetti di varia natura per arrivare a prendere dei reward o esplorare meglio l'arena.
- utilizzare le rampe per muoversi nell'arena ed esplorare l'ambiente.
- raggiungere una sfera che si trova all'interno di una sorta di labirinto, con l'agente che all'inizio vede cadere dall'alto la sfera al centro del labirinto.

Negli esempi mostrati in figura 36 vengono riportati alcuni scenari di test in più data la grande quantità di casistiche da testare.

5.2.2.4 *Categoria 4*

In questa categoria vengono introdotte le zone a tutti gli elementi già esistenti, al fine di isolare la capacità di evitare stimoli negativi si è però scelto di limitare il più possibile l'utilizzo di altre tipologie di oggetto, quando non strettamente necessarie. In modo da non includere elementi di confusione mentre si prova a verificare la capacità cognitiva ricercata. Nell'ideare questi test si è voluto verificare che l'agente sia in grado di:

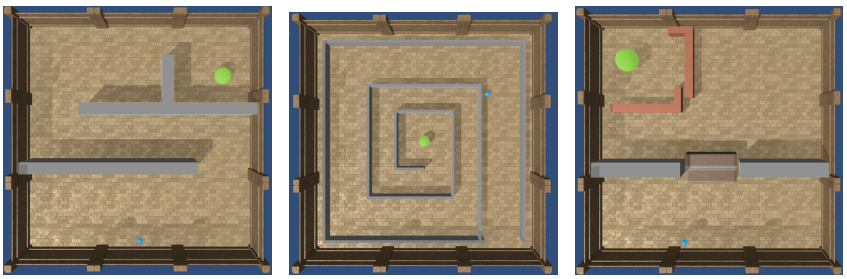
- evitare in assoluto le zone rosse, anche nel caso in cui non ci sia nessuna sfera verde/gialla nell'arena.
- riuscire a raccogliere le ricompense evitando le zone rosse poste nel suo tragitto.
- riuscire a raccogliere sfere verdi/gialle anche se vicinissime a zone rosse .
- preferire le zone arancioni a quelle rosse
- evitare le zone arancioni se per arrivare alla ricompensa esiste un altra strada sicura, anche se più lunga.



(a) *Test 1 - Raccogliere una sfera nascosta dietro un muro*

(b) *Test 9 - Raccogliere una sfera posta in alto utilizzando due rampe*

(c) *Test 10 - Utilizzare le rampe andare nella zona con la sfera altrimenti inaccessibile*



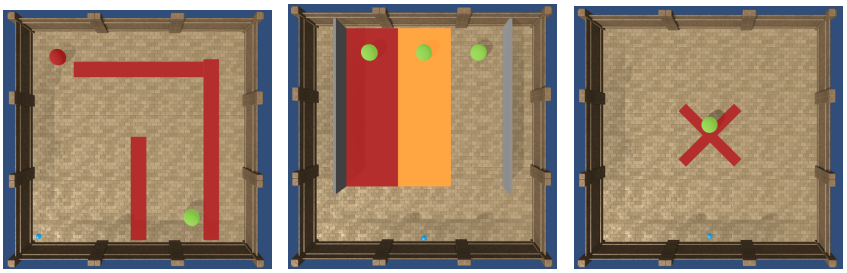
(d) *Test 12 - Raccogliere una sfera superando diversi muri che ostacolano l'agente*

(e) *Test 15 - raccogliere una sfera posta al centro di un semplice labirinto*

(f) *Test 23 - raccogliere una sfera spostando diversi tipi di oggetti.*

Figura 35: Alcuni esempi di test della categoria 3.

Dopo aver esaminato i principi che stanno dietro alla progettazione dei test locali è bene fare una precisazione. Quando si sviluppa un test cercando di dimostrare una certa proprietà cognitiva di un agente non esiste una correlazione certa fra superamento del test e possedimento della capacità in questione. Questo vale sopratutto quando si cerca di dimostrare *l'intelligenza* di un agente, essa infatti non è tanto una proprietà dell'agente, ma piuttosto risiede negli occhi di chi osserva l'agente.
Quando si osserva un comportamento bisogna ricordare che esso è il risultato dell'interazione fra ambiente e meccanismi interni dell'agente ed un comportamento che appare complesso non necessariamente indica l'esistenza di un meccanismo complesso alla base.
Questa considerazione può essere letta in due modi, il primo è che anche a partire da meccanismi semplici si possano ottenere comportamenti complessi, nella visione di Herbert Simon[40]. Il secondo è che il superamento di un test potrebbe essere frutto di circostanze casuali, senza possedere davvero una certa capacità cognitiva.



(a) *Test 5 - Raccogliere la sfera verde posta fra tre zone rosse*

(b) *Test 10 - Raccogliere la sfera verde scegliendo il percorso che non include zone arancioni, anche se più lungo*

(c) *Test 13 - Riuscire a raccogliere una sfera verde anche se posta in una posizione molto critica*

Figura 36: Alcuni esempi di test della categoria 4. Nel 2 caso il test viene superato solo se non si attraversa la zona arancio o ci si passi brevissimamente.

5.2.3 *L'algoritmo di RL: PPO come caso di studio*

Come riportato in fase di analisi visti gli obiettivi del progetto sarebbe stato troppo lungo implementare da zero un algoritmo di RL, è infatti prassi in questo campo partire da delle implementazioni solide, robuste e condivise per affrontare nuovi problemi con maggiore efficacia. Proprio per questo motivo la scelta è ricaduta sulle ottime implementazione fornite da Stable-Baseline[12], un fork di Open-AI Baselines[6].
Dell'algoritmo PPO si è parlato estensivamente in tutto il capitolo 2, in questa sezione vengono discussi aspetti perlopiù implementativi, in ogni caso si è scelto questo algoritmo in quanto rappresentante lo stato dell'arte in materia, sia in quanto ad applicabilità in vari ambiti sia riguardo ad efficacia ed efficienza. Si è innanzitutto deciso di utilizzare la versione appositamente sviluppata per l'esecuzione su GPU, detta PPO2[1], dato che l'hardware a disposizione è dotato di schede grafiche performanti.
L'utilizzo di queste baseline è molto semplice ed intuitivo, come si può notare dal codice qui riportato, tratto da [12]:

```
import gym
from stable_baselines.common.policies import MlpPolicy
from stable_baselines.common import make_vec_env
from stable_baselines import PPO2

# multiprocess environment
env = make_vec_env('CartPole-v1', n_envs=4)

model = PPO2(MlpPolicy, env, verbose=1)
model.learn(total_timesteps=25000)
```

1 La documentazione relativa è disponibile all'indirizzo https://stable-baselines.readthedocs.io/en/master/modules/ppo2.html

```
11    model.save("ppo2_cartpole")
12
13    del model # remove to demonstrate saving and loading
14
15    model = PPO2.load("ppo2_cartpole")
16
17    # Enjoy trained agent
18    obs = env.reset()
19    while True:
20        action, _states = model.predict(obs)
21        obs, rewards, dones, info = env.step(action)
22        env.render()
```

Nelle 22 righe di codice riportate sono comprese fasi fondamentali del ciclo di vita un modello di RL, ovvero la creazione dell'ambiente, definizione del modello, addestramento e salvataggio del modello, caricamento del modello e suo utilizzo in un ambiente. Questa sua semplicità ci permette di concentrarci sugli aspetti più legati al contesto applicativo, affidandosi ad algoritmi condivisi ed ottimizzati dalla comunità.
Per sviluppare un agente che possieda la capacità cognitive richieste si interverrà quindi su vari aspetti, quali la natura dell'observation, la specificazione di apposite funzioni di reward, la definizione dell'architettura di rete che utilizza l'algoritmo.
Di seguito verranno specificati i parametri più importanti quando si istanzia un modello PPO2 per l'addestramento:

- **policy**: Il modello di policy da utilizzare (MlpPolicy, CnnPolicy, CnnLstmPolicy, ...). In fase di configurazione dell'esperimento verrà richiesto come strutturare la rete della policy.
- **env** – L'ambiente da utilizzare per l'addestramento, nella prossima sezione verrà spiegato come adattare l'ambiente della Animal-AI Olympics.
- **gamma** – Discount factor
- **n_steps** – Il numero di step da eseguire per ogni environment ad ogni aggiornamento.
- **ent_coef** – Il coefficiente di entropia nel calcolo della loss.
- **learning_rate** - Il classico learning rate
- **vf_coef** – Il coefficiente per la Value function nel calcolo della loss.
- **max_grad_norm** –Il valore massimo per il clipping del gradiente
- **nminibatches** – Il numero di minibatch per ogni update, se si utilizzano policy ricorrenti il numero degli ambienti eseguiti in parallelo deve essere un multiplo di questo parametro.

- **noptepochs** – Numero di epoche per cui ottimizzare il surrogato
- **cliprange** – Il parametro di clipping
- **cliprange_vf** – Il parametro di clipping per la value function, esso non è presente nella versione originale di PPO.

5.2.4 *Infrastruttura di training*

Dopo aver introdotto l'infrastruttura di test e l'algoritmo di RL che verrà incorporato nell'agente, in questa sezione si farà menzione degli aspetti più importanti dell'infrastruttura dedicata all'addestramento.

5.2.4.1 *Configurazione addestramento*

Come già discusso, la possibilità di lanciare esperimenti con appositi script con cui configurare l'addestramento è fondamentale nel campo del ML, in quanto molto spesso si vuole far eseguire diversi esperimenti senza intervenire direttamente, dato che solitamente si tratta di esecuzioni molto lunghe e perciò è bene non sprecare tempo prezioso.
Il file principale tramite il quale avviare un esperimento, denominato "*trainPPO2.py*", contiene le definizioni di tutti i possibili parametri di configurazione utilizzabili. Per raccogliere questi valori si è utilizzato il classico modulo di Python chiamato *argparse*, a titolo dimostrativo viene riportato un estratto della definizione dei possibili parametri:

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument('--policyName',type=str,default='
    MlpLstmPolicy')
parser.add_argument('--allSharedLayers',type=str2bool,
    default=False)
parser.add_argument('--vf_layers', nargs='+', type=int,
    default=[64])
parser.add_argument('--pi_layers', nargs='+', type=int,
    default=[64])
parser.add_argument('--rewardFunName',type=str,default=None
    )
parser.add_argument('--basepath',type=str,default=basepath)
parser.add_argument('--savepath',type=str,default=savepath)
parser.add_argument('--initial_training_categories', nargs=
    '+', type=int, default=[1,2,4])
parser.add_argument('--n_epochs_easy', type=int, default=5)
parser.add_argument('--training_categories', nargs='+',
    type=int, default=[1,2,3,4,5])
parser.add_argument('--validation_frequency', type=int,
    default=5)
...

```

Si sono quindi utilizzati degli script bash come il seguente:

```
#!/bin/bash
```

```
python /aaio/animalai/trainPPO2.py \
    --policyName='MlpPolicy' \
    --allSharedLayers=False \
    --vf_layers 156 \
    --pi_layers 156 \
    --rewardFunName='AngleAndVelocityRewardFunBisProva' \
    --basepath='/aaio/animalai/' \
    --savepath='/aaio/animalai/models/' \
    --initial_training_categories 1 2 4 \
    --training_categories 1 2 3 4 \
    --n_epochs_easy=12\
    --validation_frequency=1
...
```

5.2.4.2 *Adattamento dell'ambiente della competizione a PPO*

Come si è riportato in precedenza, PPO2 è la versione di PPO implementata appositamente per l'utilizzo con GPU e fa utilizzo di ambienti vettorializzati[2], essi infatti permettono di addestrare un agente su n ambienti per step. Il nome deriva dal fatto che n ambienti indipendenti vengono messi accodati come se fossero un unico environment. In questo caso è utilizzata l'incarnazione denominata *SubprocVecEnv*, la quale crea un processo per ogni ambiente, così da velocizzare enormemente l'esecuzione dello stesso, sopratutto in quei casi, come il nostro, dove l'ambiente è computazionalmente complesso.
Per istanziare un *SubprocVecEnv* bisogna fornire una lista di funzioni che restituiscono l'ambiente da creare nei sub-processi (alcuni parametri sono relativi all'approccio seguito, del quali si parlerà nella prossima sezione):

```
env = SubprocVecEnv([lambda: getGymEnv(arena_configs=
    arena_configs_in,docker_train=docker_train,   basepath=
    basepath, wrapper=wrapper, colorCounter=colorCounter,
    categories=initial_traininig_categories, rewardFunName=
    rewardFunName) for i in range(n_cpu)])
```

La funzione *getGymEnv* si occupa proprio di avviare l'ambiente della competizione Animal-AI Olympics(AAIO) tramite il wrapper Gym messo a disposizione. A questo punto entra in gioco un aspetto importante, ovvero il far funzionare il wrapper Gym della competizione con PPO2, purtroppo infatti l'implementazione originale presentava alcuni problemi nella specifica degli spazi delle osservazioni e delle azioni, il che non ne permetteva l'utilizzo diretto.
Per superare questo problema si è quindi sfruttata la possibilità di definire degli appositi wrapper gym in cui includere quello originale della competizione e superando le problematiche senza dover cambiare il codice originale.
A titolo esemplificativo viene riportato il wrapper definito per utilizzare

2 Documentazione alla pagina: https://stable-baselines.readthedocs.io/en/master/guide/vec_envs.html

l'ambiente AAIO utilizzando solo le informazioni visuali, infatti l'implementazione PPO2, come la maggior parte delle implementazioni, non può gestire due tipi di observation con dimensioni diverse, nel caso si vogliano utilizzare entrambe bisognerà fare delle variazioni.

```
class OnlyVisualObservationWrapper(Wrapper):
    def __init__(self, env,configs,categories,
    rewardFunction):
        super(OnlyVisualObservationWrapper, self).__init__(
    env)
        self.arena_configs_in= configs
        self.rewardFunction = rewardFunction
        self.observation_space = Box(low=0, high=255, shape
    =(84,84,3), dtype=np.uint8)
        self.categories = categories

    def reset(self, **kwargs):
        c_ind = self.categories[random.randint(0, len(self.
    categories)-1)]
        t_ind = random.randint(0, len(self.arena_configs_in
    [c_ind]) - 1)
        obs = self.env.reset(self.arena_configs_in[c_ind][
    t_ind])
        return obs[0]

    def changeCategories(self,categories):
        self.categories = categories

    def step(self, action):
        obs, r, d, _ = self.env.step(action)
        visual = obs[0]
        reward = self.rewardFunction(r,obs)
        done = d
        return visual,reward,done,_
```

Come si evince dal codice riportato, un wrapper deve definire i classici metodi *init(),step(), reset()* e specificare qual'è l'ambiente Gym da incapsulare. Oltre a questi metodi è stato definito un metodo per cambiare le categorie su cui fare addestramento, infatti ad ogni reset dell'ambiente viene scelta una configurazione dell'arena tra quelle disponibili appartenenti alle categorie su cui si è scelto di addestrare l'agente. Il perché del cambio di categorie durante l'addestramento verrà descritto nella prossima sezione quando si parlerà di curriculum learning.
Aggiungere metodi rispetto a quelli standard non è banale nel caso di ambiente vettorizzati multiprocesso, in quanto per chiamare un metodo dovrà essere eseguita una comunicazione tra processi, ciò ha quindi richiesto alcune piccole modifiche al codice di *SubprocVecEnv*.
La definizione di Wrapper è un meccanismo molto potente, in quanto permette di modificare tantissimi aspetti senza dover toccare l'algoritmo di RL che si sta utilizzando. E' infatti possibile modificare l'observation dell'agente prima di sottoporla all'algoritmo, si può modificare l'attribuzione di reward e tante altre cose, il tutto dovendo soltanto rispettare l'interfaccia dei Wrapper Gym.

Proprio per questo durante lo sviluppo del progetto sono stati implementati diversi Wrapper, in base all'approccio adottato.
L'infrastruttura di addestramento è stata pensata in modo che per cambiare il tipo di wrapper da utilizzare sia sufficiente specificare il nome dello stesso in fase di configurazione ed in automatico venga recuperato ed istanziato il giusto wrapper.

5.2.4.3 *Il ciclo d'addestramento*

Dopo aver introdotto alcuni aspetti dell'infrastruttura di addestramento verranno ora elencate le possibili fasi in esso:

- Dopo aver caricato la configurazione dell'addestramento viene creata un'apposita cartella dove salvare tutti i dati relativi a questo specifico addestramento.
- Il primo file ad essere salvato è proprio un JSON contenente tutti i parametri di configurazione che saranno utilizzati nell'addestramento, in modo da poter disporre di tutte le informazioni in un momento successivo, anche se lo script bash da cui è stato lanciato l'esperimento è stato modificato.
- Se richiesto, viene caricato un modello addestrato in precedenza, altrimenti viene creata un istanza di PPO2 specificando i parametri scelti.
- In fase di configurazione dovrà essere specificato ogni quante iterazioni di aggiornamento salvare il modello (parametro *save_freq*), informalmente si chiamerà questo periodo epoca. Il numero di epoche (da non confondere con il numero di epoche per cui ottimizzare il surrogato di PPO in un singolo passo di update) di addestramento deve essere anch'esso specificato (parametro *epochs*). Per cui il totale di passi d'aggiornamento è dato da $save\_freq \times epochs$.
- Contestualmente all'avvio dell'addestramento verrà creato un file contenente i risultati delle validazioni intermedie ed uno relativo all'andamento dell'addestramento per ogni passo di aggiornamento.
- L'addestramento verrà eseguito nell'ambiente della competizione, utilizzando come configurazioni dell'arena quelle appositamente predisposte. In fase di configurazione vanno specificate le categorie in cui addestrare l'agente, questa scelta può essere rivista dopo un numero a piacere di epoche, tramite il solito file di configurazione.
- Ogni un certo numero di epoche (al solito specificate) si può effettuare una sorta di "validazione" dell'agente, ovvero esso viene testato su tutte le categorie scelte, nei task appositamente specificati. Il risultato per ogni categoria affrontata di queste

validazioni andrà in un file complessivo, creato all'inizio, mentre il risultato dettagliato, task per task andrà in uno specifico file creato appositamente. L'idea è di avere a disposizione più dati possibili per successive analisi.

5.2.5 *Configurazioni ambienti per training*

Come descritto poc'anzi delle specifiche configurazioni dell'arena sono state utilizzate per l'addestramento dell'agente. In questo caso si è seguito un approccio diverso rispetto a quello della progettazione delle configurazioni per i test dell'agente. Infatti mentre nei test bisogna codificare situazioni ben precise e valutabili, con una randomicità, che se presente, deve essere controllata e riproducibile, nell'addestramento l'obiettivo deve essere quello di far "vivere" le situazione più varie possibili all'agente.
La capacità di generalizzare è proprio l'obiettivo di qualsiasi algoritmo di ML, un algoritmo che si comporta bene soltanto su i dati di addestramento è, di fatto, inutile. Dei buoni e variegati dati per l'addestramento sono sicuramente un ottima base per costruire un agente che rispetti le attese.
Fatta questa doverosa premessa, in questo caso la varietà dei dati di addestramento è stata ottenuta sfruttando la possibilità di non specificare la posizione o la dimensione di certi oggetti, lasciando al caso questi valori. Così facendo l'agente si troverà davanti ad un grandissimo numero di situazioni diverse, e disponendo perciò di più dati per generalizzare.
A titolo esemplificativo nelle figure 37, 38, 39 e 40, sono riportate alcune variazioni a partire dallo stesso file di configurazione.
In questo caso non si sono poste soglie di punteggio da superare, in quanto l'unico obbiettivo in fase di addestramento è quello di massimizzare il punteggio da ottenere.

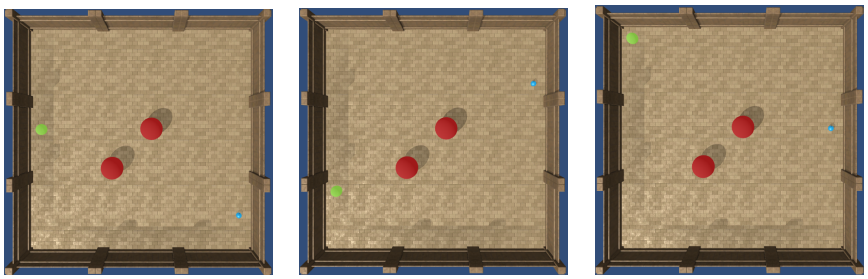


Figura 37: Alcuni esempi di varianti dello stesso file di configurazione per l'addestramento relativo alla categoria 1

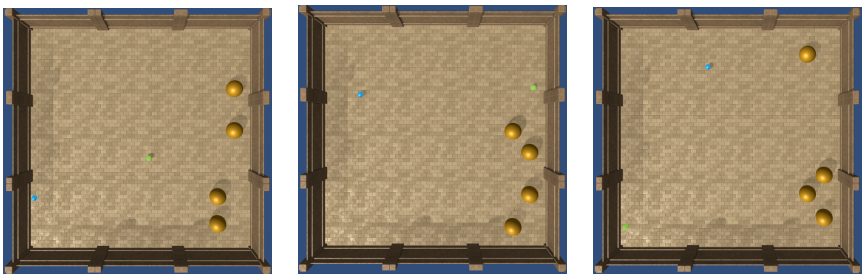


Figura 38: Alcuni esempi di varianti dello stesso file di configurazione per l'addestramento relativo alla categoria 2

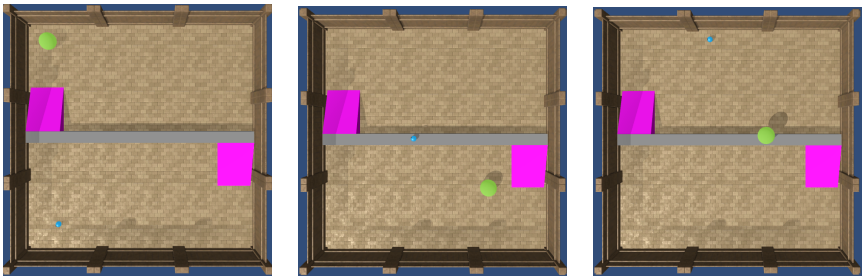


Figura 39: Alcuni esempi di varianti dello stesso file di configurazione per l'addestramento relativo alla categoria 3

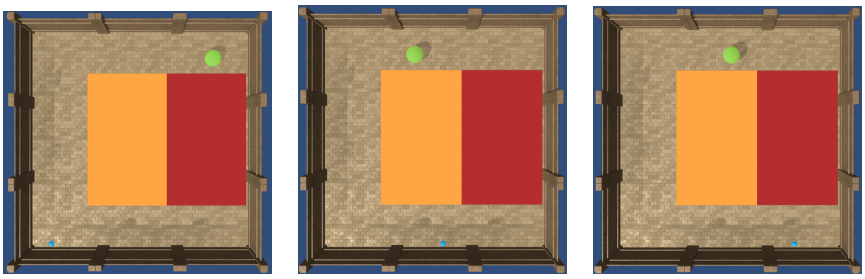


Figura 40: Alcuni esempi di varianti dello stesso file di configurazione per l'addestramento relativo alla categoria 4

5.3 MODELLARE ED ADDESTRARE UN AGENTE

In questa sezione verranno finalmente illustrate le modalità e gli approcci utilizzati per affrontare al meglio delle possibilità la competizione Animal-AI Olympics. Si precisa che l'ordine con cui verranno presentati non è detto coincida con quello effettivo di implementazione ed utilizzo, infatti alcuni di questi approcci sono stati utilizzati insieme per gran parte del tempo. Ciò che si vuole proporre è solamente un ordine concettuale.

5.3.1 *Limiti dell'utilizzo di immagini RGB pure*

Come estensivamente trattato nella sezione 3.3.3, l'agente riceve dall'ambiente un immagine rappresentante ciò che vede ed un vettore contenente le velocità percepite su i tre assi.
La prima idea e più semplice idea che può venire in mente è quella di utilizzare direttamente l'immagini raccolte per addestrare l'algoritmo di RL, per cui come input esso riceverà un vettore di dimensioni [84,84,3] e riceverà punteggi/penalità secondo le modalità previste dall'ambiente. Quest'approccio si è pero rivelato poco efficace, come indicato sia dagli organizzatori della competizione, sia da test svolti nell'ambito di questo progetto. Gli organizzatori hanno infatti, al momento del lancio della competizione, messo a disposizione i risultati ottenuti applicando un algoritmo standard come PPO, fornendo di fatto delle baseline con cui confrontarsi.[3]

Agente	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	Totale
Hand-Coded	21	15	0	8	3	15	12	0	11	0	28.33
PPO-Obstacle	18	13	4	7	2	5	9	2	4	0	21.33
PPO-Spatial	10	6	3	3	1	5	7	1	6	2	14.67
PPO-Food	5	4	4	3	0	2	2	2	0	1	7.67
Up(.8),Left(.2)	4	0	0	2	0	5	5	0	0	0	5.33
Random	2	0	0	3	0	0	0	0	0	0	1.67

Tabella 3: Risultati di baseline fornite dagli organizzatori della competizione. Con Up(.8), Left(.2) si intende che l'agente si intende che con una probabilità di 0.8 viene scelto di far andare avanti l'agente e con 0.2 di ruotare a sinistra. Per hand-coded si intende un agente programmato secondo un euristica definita dallo sviluppatore. I tre agenti che iniziano per PPO sono stati addestrati con l'algoritmo omonimo su tre configurazione di esempio fornite nella competizione.

Come si evince dalla tabella 3, il risultato migliore raggiunto da PPO

3 Risultati e dettagli all'indirizzo https://www.mdcrosby.com/blog/animalaihand.html

è 21.33, ottenuto addestrando l'agente sulla configurazione di esempio [4] con per la terza categoria, contenente due muri ed una sfera verde. Gli altri due risultati ottenuti con PPO sono ancora più eloquenti, ottenendo dei punteggi veramente bassi, sopratutto nel caso PPO-Food, addestrato in un arena contenente una sola sfera verde. Deve far riflettere il fatto che uno scenario così semplice, con un arena completamente vuota ed una sola sfera verde non riesca ad essere appreso eseguito con successo, ottenendo risultati paragonabili a quelli di una strategia pseudo casuale quale la penultima della tabella. Si precisa che i test in questione sono stati eseguiti dagli organizzatori della competizione utilizzando il toolkit Unity ML-Agents.
Successivi test, realizzati tramite l'ausilio della piattaforma di addestramento sviluppata in questo progetto hanno confermato gli esiti riportati dagli organizzatori.
L'inefficacia dell'utilizzo delle pure immagini RGB potrebbe risiedere in vari aspetti, tra i quali si può sicuramente annoverare la sparsità dei reward, la durata molto breve degli episodi e la dimensionalità dell'input.
Dato che molto spesso quando si utilizzano algoritmi di RL è bene utilizzare anche una CNN per l'estrazione di feature dalle immagini, si è testato se il loro utilizzo potesse risolvere la situazione, ma nemmeno in questo caso si sono ottenuti risultati soddisfacenti. Bisogna però specificare che il risultato potrebbe essere frutto di un errato dimensionamento della rete che descrive la policy o di altri iper-parametri, i risultati sono stati ottenuti senza ricerca degli iper-parametri migliori, lasciando quelli standard.

5.3.2 *Semplificare il modello considerando solo elementi di interesse*

Dato che l'utilizzo delle immagini RGB di partenza, senza nessun altro tipo di strategia, è inefficace, si è pensato a come poter semplificare il problema al fine di ottenere migliori risultati.
Se si analizza il problema ci si accorge che prima di qualsiasi cosa l'agente deve possedere degli stimoli rispetto agli elementi fondamentali inclusi in tutti i test della competizione. Questi elementi principali sono ovviamente le sfere rosse, verdi e gialle e le zone rosse ed arancioni, esse le troviamo invariate nella loro sostanza in ogni situazione.
Un agente dovrà quindi necessariamente associare uno stimolo di attrazione verso le sfere gialle e verdi ed al contempo una sorta di repulsione verso le sfere rosse e verso le zone mortali/pericolose.
L'agente dovrà poi collegare questi stimoli alle azioni per perseguirli, ma intanto il primo passo è dare importanza a questi stimoli.
Quindi l'intuizione è stata quella di far percepire all'agente solo queste informazioni fondamentali, che tradotto in codice, equivale ad avere un

4 Disponibile all'indirizzo: https://github.com/beyretb/AnimalAI-Olympics/blob/master/examples/configs/3-Obstacles.yaml

canale per ognuno dei 4 colori fondamentali per questi stimoli, ovvero il verde, il giallo, il rosso e l'arancione. Quindi ricapitolando si otterrà una matrice delle stesse dimensioni dell'immagine di partenza (84x84), con al posto dei tre canali dei colori RGB, un canale per ogni colore che indica se in una determinata cella dell'immagine di partenza è presente o meno quel determinato colore.
Proprio a questo scopo è stata implementata una classe di utility detta *ColorCounter*, la quale specificati i colori da ricercare e l'immagine di partenza, restituisce un vettore con le proprietà descritte. Il metodo per cercare un determinato colore in una matrice di pixel è il seguente:

```
def _get_colours(self,pixels, colour):
    return np.all(pixels > np.minimum(colour * .8, colour -
     25), axis=-1) & np.all(
        pixels < np.maximum(colour * 1.2, colour + 25),
    axis=-1)
```

Per costruire il vettore da dare in input all'agente invece:

```
def getState(self,visualObs,vectorObs=None,action=None):
    allPoints = np.zeros(shape=(84,84,len(self.colors)))
    obs = visualObs*255
    for i in range(0,len(self.colors)):
        allPoints[:,:,i] = self._get_colours(obs,self.
    colors[i])
    return allPoints
```

Questo semplice ma potente accorgimento ha prodotto dei risultati notevoli, dimostrando la bontà dell'approccio. Nella tabella 4 si può notare come questo approccio da solo porti a superare il punteggio ottenuto dall'algoritmo scritto a mano di baseline. Nel caso in tabella l'agente è stato addestrato per 11 Milioni e 264 mila step, considerando solamente i colori verde, giallo e rosso.
Ovviamente questo metodo è limitato, dato che non si considerano di fatto tutti gli altri oggetti presenti nell'arena, però può costituire un buon nucleo di base, una sorta di elemento istintivo dell'agente. L'agente durante l'addestramento avrà quindi imparato ad associare le zone con molti punti rossi a zone da dove fuggire, mentre zone con punti gialli/verdi zone verso cui dirigersi.

Test	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	Totale
Ufficiali	23	16	6	8	3	13	10	3	9	1	30,67
Hand-coded	21	15	0	8	3	15	12	0	11	0	28.33
Locali	28	23	2	11	ND	ND	ND	ND	ND	ND	21.33
PPO-Obstacles	18	13	4	7	2	5	9	2	4	0	21.33

Tabella 4: Risultati ottenuti nei test ufficiali ed in quelli locali da un agente addestrato con il metodo descritto per 11 Milioni e 264 mila step di addestramento, messi a confronto con il miglior risultato ottenuto da PPO nelle baseline e con l'euristica. I test per ogni categoria sono 30, i test locali sono stati condotti solo per le prime 4 categorie

Il concetto in questo caso è stato applicato soltanto ai colori legati agli stimoli più diretti, ma può essere applicato anche agli altri colori caratteristici degli oggetti, quali il grigio per i muri, il blu per le piattaforme ed il rosa per le rampe. Il metodo perde un po' di significato se si considerano troppi colori, in quanto si va a creare un input molto rumoroso, da cui sarebbe difficile addestrare un buon modello.

5.3.3 *Curriculum Learning*

> *Humans and animals learn much better when the examples are not randomly presented but organized in a meaningful order which illustrates gradually more concepts, and gradually more complex ones.*[2]

Queste righe, tratte dall'articolo *Curriculum Learning*[2], racchiudono un'idea molto potente, che sta alla base dell'apprendimento umano ed animale, ovvero il presentare gradualmente concetti via via più complessi ma partendo da esempi più semplici.
Questo approccio si è dimostrato efficace in molto contesti applicativi e si è perciò deciso di applicarla per affrontare l'addestramento.
In questo specifico contesto si è declinato il concetto iniziando l'addestramento con alcune categorie, per poi introdurne altre solo successivamente.
A prescindere dagli approcci utilizzati si è scelto di adottare sempre la seguente strategia:

- Iniziare l'addestramento utilizzando le configurazioni di addestramento progettate per le categorie 1, 2 e 4. Quindi di fatto introducendo solo le categorie dove sono presenti gli stimoli principali di repulsione ed attrazione.

- Introdurre dopo un certo numero di epoche, tipicamente 12, l'addestramento negli scenari della categoria 3.

L'idea che sta dietro questa divisione è quella di dare il tempo all'agente di apprendere la relazione fra i vari oggetti che danno ricompense positive e quelli che invece infliggono penalità, per poi introdurre quegli elementi di complicazione costituiti da rampe, muri e quant'altro.
Per specificare questi aspetti si sono predisposti i seguenti parametri nel file di configurazione:

```
...
  --initial_training_categories 1 2 4 \
  --training_categories 1 2 3 4 \
  --n_epochs_easy=12\
...
```

5.3.4 *Dividere l'osservazione visuale in quadranti*

A partire dall'idea esplorata nella sezione 5.3.2 si è pensato come semplificare ancora di più l'input da passare all'algoritmo di RL. Infatti pur includendo nell'observation soltanto gli elementi di interesse, gli elementi da considerare sono comunque tanti, in quanto la matrice di input, se si utilizzano i 4 soliti colori, conterrà 28224 elementi.
Dato che di fatto quello che si vuole implementare è una sorta di stimolo, non è così necessario comunicare all'agente pixel per pixel quali colori contiene l'immagine, anche considerando che comunque gli oggetti si estendono per un certo gruppo di pixel. Potrebbe essere quindi sufficiente dare un indicazione più sommaria della distribuzione dei colori verde, giallo, arancione nell'immagine acquisita, così da dare all'agente un indicazione già più discretizzata ed utilizzabile. Per farlo si è pensato

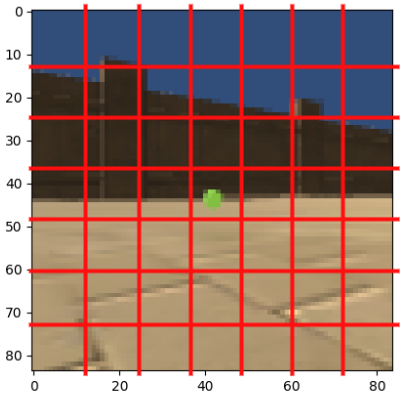


Figura 41: Esempio di immagine catturata dall'agente divisa in 49 quadranti

di dividere l'immagine originaria in quadranti, come mostrato in figura 41, adottando il seguente procedimento:

- Si divide l'observation dell'agente in un certo numero di quadranti.
- Per ognuno di essi si calcola il numero di punti verdi (reward), gialli (altro reward), rossi (penalità).
- Si fornisce a PPO un vettore lungo X * Y , con X numero di quadranti considerati e Y numero colori considerati, contenente le informazioni calcolate.

Un aspetto da valutare è in quanti quadranti dividere l'immagine, infatti se si opera una divisione in pochi quadranti si rischia di fornire all'algoritmo informazioni troppo sommarie, se invece si divide in troppo quadranti ci si riconduce al caso precedente, con informazioni troppo rumorose. Facendo vari esperimenti è risultato empiricamente che la scelta migliore è dividere in 49 quadranti, il che rappresenta un buon trade-off fra i due estremi.

L'agente ricevendo queste informazioni imparerà ad eseguire le azioni giuste in base al contenuto dei vari quadranti.
Per implementare questo approccio è stato esteso il metodo presentato in precedenza, aggiungendo uno specifico metodo per il calcolo dei punti nei quadranti:

```
def _getCounts(self,obs,color):
    values = self._get_colours(obs * 255, color)
    color_points = []
    for i in range(self.num_rows):
        r_start_index = i*self.row_len
        r_end_index = (i+1)*self.row_len
        for j in range(self.num_column):
            c_start_index = j * self.column_len
            c_end_index = (j + 1) * self.column_len
            color_points.append(values[r_start_index:
    r_end_index,c_start_index:c_end_index].sum())
    return np.reshape(color_points, (1, self.num_quadrant))


def getState(self,visualObs,vectorObs=None,action=None):
    self.last_speed = vectorObs
    if self.keep_points:
        allPoints = np.zeros(shape=(84,84,len(self.colors))
    )
        obs = visualObs*255
        for i in range(0,len(self.colors)):
            allPoints[:,:,i] = self._get_colours(obs,self.
    colors[i])
        return allPoints
    else:
        firstCounts = self._getCounts(visualObs,self.colors
    [0])
        allPoints = firstCounts
        for i in range(1,len(self.colors)):
            counts = self._getCounts(visualObs,self.colors[
    i])
            allPoints =np.concatenate((allPoints,counts),
    axis=None)
        if(self.considerVelocities):
            allPoints = np.concatenate((allPoints,vectorObs
    ), axis= None)
        return allPoints
```

Questo approccio seppure trova la sua naturale applicazione con i 4 colori di premi e penalità, può essere utilizzato anche con gli altri colori degli oggetti, ovviamente con le stesse considerazione fatte nella sezione precedente.
Ovviamente se si considerano solo i colori elementari si riusciranno a superare solo pochi test relativi alla categoria 3, ma la vera questione è se, includendo ad esempio i colori grigio, blu e rosa i risultati su questa categoria migliorino. Evidenze nei risultati sembrano indicare che sia effettivamente utile includere questi colori per quanto riguarda la terza categoria, a titolo esemplificativo vengono confrontati nella tabella 5 i risultati ufficiali ottenuti considerando solo i colori verde, giallo e rosso

oppure aggiungendo gli altri 3.

Test	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	Totale
GYR	20	19	5	11	5	11	14	4	7	1	32,33
GYRGPB	19	7	8	12	4	12	8	0	1	0	23,67

Tabella 5: Risultati ottenuti nei test ufficiali considerando i colori verde, giallo e rosso (GYR) e considerando i precedenti tre più il grigio, il rosa ed il blu (GYRGPB).

Come si può notare il primo metodo è più efficace in generale, ma il secondo riesce a fare bene nella terza categoria, anche considerando che il miglior risultato ottenuto in tutta la competizione per questa categoria è di 12 episodi superati, il che non è troppo lontano dagli 8 di questo approccio.

5.3.5 *Utilizzare la velocità dell'agente*

Fin ora si è parlato solo degli input visivi, in realtà anche le velocità percepite dall'agente possono essere un ausilio molto utile ai fini dell'addestramento.
Le velocità percepite sui tre assi ci dicono infatti se l'agente è fermo, se sta andando dritto o se sta variando la sua altitudine nell'arena. Certe informazioni sarebbero infatti impossibili o perlomeno molto difficili da desumere solamente a partire dalle immagini.
Si pensi, ad esempio, ad un caso in cui l'agente sta "spingendo" una scatola, di fatto l'immagine percepita sarà sempre la stessa se la scatola occupa l'intero campo visivo dell'agente; utilizzando solo delle immagini l'agente non potrà capire che si sta muovendo, mentre con il vettore delle velocità percepite questa informazione è subito disponibile.
Oltre a questi casi, la velocità può essere anche molto utile per percepire contatti con degli oggetti, se si volesse infatti considerare soltanto i soliti colori dei reward, nessun altro tipo di oggetto verrebbe percepito e perciò l'agente potrebbe anche essere ostacolato da un muro senza che esso se ne accorga. Ma se si ha a disposizione la velocità si potrà percepire immediatamente di essere incorsi in una collisione, motivo per cui l'agente sceglierà delle azioni alternative.
Operativamente le tre velocità percepite sono state quindi aggiunte al vettore formato formato come specificato nella sezione 5.3.4, la cui lunghezza è quindi ora di $X \times Y + 3$.
L'utilizzo dei wrapper era stato già introdotto nella sezione 5.2.4.2, in questo caso se ne è definito uno apposito per poter utilizzare insieme i due approcci:

```
maxSpeed = 22
class SimplifiedWrapper(Wrapper):
    def __init__(self, env, configs,colorCounter,categories
    ):
```

```
        super(SimplifiedWrapper, self).__init__(env)
        self.arena_configs_in = configs
        self.colorCounter = colorCounter
        self.obs_vector_length = self.colorCounter.
    getStateVectorLen()
        self.last_speed = [0,0,0]
        high = 84*84 / self.colorCounter.num_quadrant
        low = 0
        if self.colorCounter.keep_points:
            high = 255
        else:
            if colorCounter.considerVelocities:
                low = -maxSpeed
            if colorCounter.considerVelocities and high <
    maxSpeed:
                high = maxSpeed
        self.observation_space = Box(low=low,high=high,
    shape=self.obs_vector_length,dtype=np.uint32)
        self.categories = categories

    def changeCategories(self,categories):
        self.categories = categories

    def reset(self, **kwargs):
        c_ind = self.categories[random.randint(0, len(self.
    categories)-1)]
        t_ind = random.randint(0, len(self.arena_configs_in
    [c_ind]) - 1)
        obs = self.env.reset(self.arena_configs_in[c_ind][
    t_ind])
        self.last_speed = [0,0,0]
        return self._getObs(obs)

    def step(self, action):
        obs, r, d, _ = self.env.step(action)
        reward = r
        done = d
        return self._getObs(obs), reward, done, _

    def _getObs(self,obs,action=None,map=None):
        visual = obs[0]
        vector = np.round(obs[1],2)
        self.last_speed = vector
        return self.colorCounter.getState(visual,vector)
```

5.3.6 *Reward Shaping*

Unitamente ai due approcci precedenti, quella che sta per essere descritto è stata un idea fondamentale nell'affrontare le sfide poste dalla competizione.
L'approccio in questione è quello del reward shaping, ovvero fornire dei segnali di reward addizionali per spingere l'agente verso comportamenti desiderati. Questo approccio può risultare molto utile sia nel accelerare l'addestramento che nel migliorare complessivamente i risultati.

Si è perciò deciso di utilizzare questo approccio praticamente in tutti gli esperimenti fatti, cercando di progettare la miglior funzione di reward possibile.
Per progettare una funzione di reward efficace si è prima osservato il comportamento degli agenti addestrati senza nessun segnale di ricompensa addizionale per poter così andare a penalizzare comportamenti considerati non corretti e premiare quelli invece vantaggiosi.
Ciò che si vedeva osservando l'agente è che, spesso, se non aveva nessuno stimolo presente nel suo campo visivo esso tendeva a girare su se stesso all'infinito. Ovviamente questo comportamento ha un senso, in quanto fare un giro su se stesso permette di vedere tutto ciò che circonda l'agente, ma nel caso in cui esso sia effettivamente lontano da oggetti significativi o in una specifica posizione nel quale essi non sono visibili, l'agente continuerà a girare su se stesso sperando di individuare qualcosa.
In generale ciò che si osservava era una tendenza a rimanere nella stessa posizione quando si affrontavano situazioni leggermente più complesse, in cui l'obiettivo non era nelle immediate vicinanze. Questo comportamento che si è sviluppato durante l'addestramento può avere una spiegazione nel fatto che rimanendo fermo non rischia di incappare in situazioni di penalità e per cui l'algoritmo ha di fatto trovato un soluzione locale che complessivamente non è troppo penalizzante.
Ovviamente questo comportamento non è desiderabile al fine di superare i test della competizione, per cui si è intervenuto per spingere l'agente a muoversi maggiormente.
Quando si progetta una funzione di reward bisogna stare molto attenti nel considerare le implicazioni delle proprie scelte progettuali, infatti le tecniche di reinforcement learning fanno di tutto per massimizzare il punteggio ottenuto e ciò potrebbe portare a trovare delle "scorciatoie" che non rispecchiano però il nostro desiderio originale.
Fatta questa doverosa premessa, qualitativamente si voleva fare in modo che l'agente mantenesse una traiettoria perlopiù rettilinea, in modo da esplorare maggiormente l'ambiente. Seguendo questa traiettoria tendenzialmente rettilinea si desidera però che la velocità sia il più alta possibile in valore assoluto.
Verrà ora presentata la funzione di reward che si è rivelata migliore in termini di risultati ottenuti, in ogni caso le altre progettate sono solo piccole variazione di quella presentata.
La funzione fa uso di due informazioni fondamentali:

- Il modulo della velocità risultante dalla somma della velocità sull'asse x e sull'asse z, calcolata come:

$$v_{xz} = \sqrt{v_x^2 + v_y^2}$$

- L'angolo α del vettore velocità risultante:

$$\alpha = \arctan 2(v_z, v_x)$$

Che vengono utilizzate nel seguente codice:

```
def computeAngle(velocities):
    return np.degrees(np.arctan2(velocities[2],velocities
    [0]))%180.0
def rewardFunction(reward, obs):
    addictionalReward = 0
    if reward < 0:
        velocities = np.round(obs[1], 2)
        # print("Velocities",velocities)
        velocitiesSquared = np.square(velocities)
        resultantSpeed = np.sqrt(velocitiesSquared[0]+
    velocitiesSquared[2])
        if resultantSpeed < 3:
            if resultantSpeed == 0:
                addictionalReward += reward
            else:
                addictionalReward += reward * 0.5
        else:
            angle = computeAngle(obs[1])
            diff = np.abs(angle -90)
            threshold = 40
            if(diff <=threshold):
                addictionalReward = (threshold+1-diff)/
    threshold * -reward*1.5
    return reward + addictionalReward
```

Come si evince dal codice sopra riportato il calcolo della funzione di reward addizionale procede così:

- Innanzitutto il calcolo viene applicato solo se il reward è negativo, ovvero se si è fatto uno step senza raccogliere nulla. Non si vogliono infatti influenzare quelle situazioni dove è stato raccolto un punteggio positivo.

- La prima cosa ad essere considerata è il modulo della velocità risultante dalla somma della velocità sull'asse x e sull'asse z, se essa è più bassa di un certo valore (in questo caso 3) si riceve una penalità secondo due casi:

  - Se la velocità è pari a zero si infligge una ulteriore penalità pari a quella già inflitta dall'ambiente[5].

  - Altrimenti si infligge una ulteriore penalità pari alla metà quella già inflitta dall'ambiente.

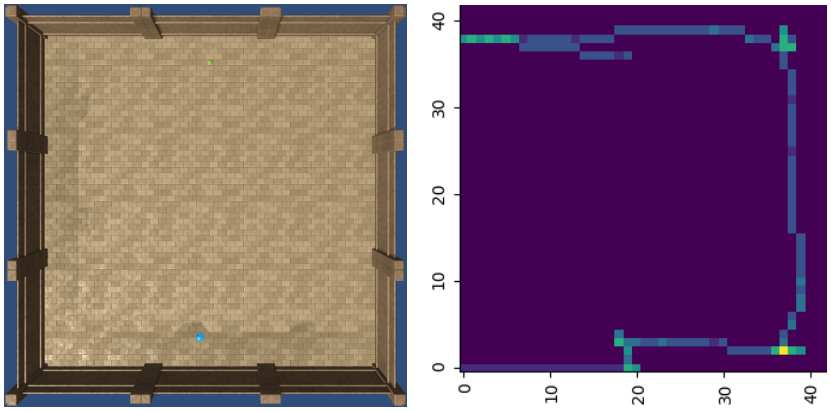
  - Dopo aver calcolato l'angolo del vettore velocità risultante si premia l'agente la sua differenza con l'angolo retto è minore di $40^{\cdot}$, quindi $50^{\cdot} \leq \alpha \leq 140^{\cdot}$. Il premio ricevuto è massimo se la differenza è 0 e minimo se la differenza è 40. Il premio ricevuto è compreso quindi tra $\frac{1}{40} \times -\frac{1}{T} \times 1.5$ e $\frac{41}{40} \times -\frac{1}{T} \times 1.5$

5 Si ricordi che l'ambiente ad ogni passo attribuisce una penalità pari a $1/T$ con T numero massimo di step dell'episodio

La funzione così descritta premia le traiettorie rettilinee e penalizza le basse velocità, portando ad una maggiore capacità esplorativa dell'agente. Grazie ad una buona progettazione per includere questa funzione di reward nel processo di addestramento è stato sufficiente estendere il wrapper definito nella sezione precedente:

```
class ShapedRewardWrapper(SimplifiedWrapper):
    def __init__(self, env, configs,colorCounter,categories
    ,rewardFunction):
        super(ShapedRewardWrapper, self).__init__(env,
    configs,colorCounter,categories)
        self.rewardFunction= rewardFunction

    def step(self, action):
        obs, r, d, _ = self.env.step(action)
        reward = self.rewardFunction(r,obs)
        done = d
        return self._getObs(obs), reward, done, _

    def _getObs(self,obs,action=None,map=None):
        visual = obs[0]
        vector = obs[1]
        return self.colorCounter.getState(visual,vector)
```

In fase di configurazione dell'addestramento è possibile scegliere la funzione di reward che si vuole tra quelle disponibili, richiedendo semplicemente di scrivere il nome della stessa e l'architettura utilizzerà in automatico quella indicata.
Il comportamento osservato dopo l'applicazione di questa funzione di reward rispecchia ciò che si desiderava, l'agente infatti non rimane praticamente mai fermo e descrive delle traiettorie molto curiose, adottando delle strategie che appaiono intelligenti, come ad esempio girare mentre si va indietro in modo da vedere sempre frontalmente e cambiare posizione appena ci si imbatte con un ostacolo. Una prova di questo fatto può essere vista nell'immagine 42, l'agente in questo caso non riesce a vedere direttamente la sfera piccolissima posta nell'altro lato dell'arena. L'agente a questo punto mette in campo una strategia intelligente, ovvero si gira e va all'indietro verso l'angolo destro in basso dell'arena (avendo così la fronte verso il centro dell'arena) per poi costeggiare sempre muovendosi all'indietro il perimetro dell'arena fino a raggiungere prima l'angolo il alto a destra e poi quello in alto a sinistra. Giunto in alto a sinistra ed orientato sempre verso l'interno dell'arena, l'agente ha a questo punto di fronte la sfera, ormai sufficientemente vicina per essere vista con chiarezza e perciò per essere raccolta.



(a) *Test 1 della categoria 1 - Una sfere di dimensioni 0.5, ovvero la più piccola possibile è posta al lato opposto dell'agente.*

(b) *Posizioni occupate dall'agente in tutto l'episodio.*

Figura 42: Situazione iniziale del test 1 della categoria 1 e grafico del movimento dell'agente addestrato utilizzando la funzione di reward aggiuntiva. Il test è stato superato.

5.3.7 *Utilizzo di reti ricorrenti*

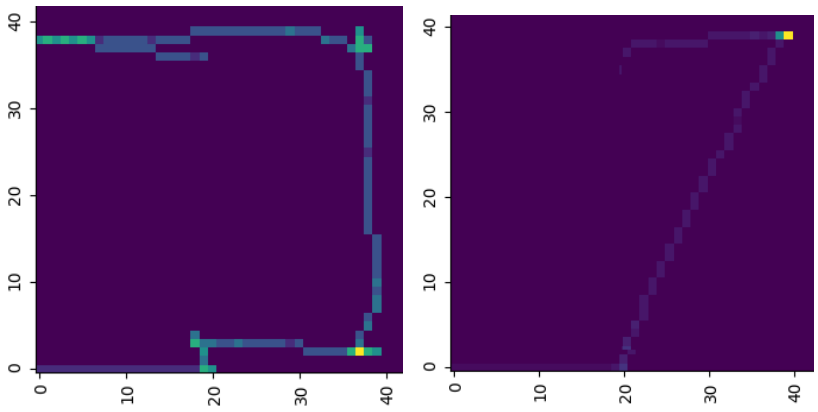
Tutti gli approcci finora esposti non fanno però uso di nessuna memoria ne modello dell'ambiente. Il che può effettivamente rappresentare un problema nei task dove potrebbe esser necessaria considerare delle informazioni provenienti dal passato.
In molti esperimenti era stata utilizzata una rete di tipo MLP feed-forward per descrivere l'architettura della policy, con un layer condiviso di lunghezza 156 per estrarre feature dall'input originale e poi due distinti strati sempre di dimensione 156 per descrivere la policy network e la value function network. Questa scelta si è dimostrata senz'altro valida, ma si è voluto testare se inserendo una rete ricorrente nell'architettura i risultati sarebbero migliorati.
Per descrivere la rete utilizzata da PPO si sono utilizzate le comode e robuste implementazioni fornite da Stable-Baselines, le quali permettono di utilizzare in modo trasparente varie tipologie di reti[6], dando anche la possibilità di specificare la dimensione dei vari livelli. Si è perciò integrata questa funzionalità nell'infrastruttura di addestramento, dando la possibilità di specificare nel file di configurazione tutte le opzioni desiderate per strutturare la rete, come illustrato nel seguente codice:

```
1 from stable_baselines.common.policies import MlpLstmPolicy,
      MlpPolicy,CnnPolicy,CnnLstmPolicy,CnnLnLstmPolicy
2 policies = {
3     'MlpPolicy': MlpPolicy,
4     'MlpLstmPolicy': MlpLstmPolicy,
5     'CnnPolicy': CnnPolicy,
```

6 Maggiori dettagli all'indirizzo https://stable-baselines.readthedocs.io/en/master/modules/policies.html

```
    'CnnLstmPolicy' : CnnLstmPolicy,
    'CnnLnLstmPolicy' : CnnLnLstmPolicy
}

def getPolicyAndKwargs(policyName,allLayerShared,
    shared_layers,useLstm,vf_layers=None,pi_layers=None):
    policy_kwargs = dict()
    if not allLayerShared and vf_layers==pi_layers and
    vf_layers == None:
        raise ValueError('If layers are not shared you must
     specify pi_layers and vf_layers')
    if allLayerShared:
        policy_kwargs = dict(net_arch=shared_layers)
        if useLstm:
            policy_kwargs['net_arch'].append('lstm')
    else:
        policy_kwargs = dict(net_arch=shared_layers)
        if useLstm:
            policy_kwargs['net_arch'].append('lstm')
        policy_kwargs['net_arch'].append(dict(vf=vf_layers,
     pi=pi_layers))

    return policies[policyName],policy_kwargs
```

L'utilizzo di una rete ricorrente condivisa fra la rete della value function e e della policy ha portato ad un miglioramento, seppur limitato, nei punteggi ottenuti nei test della competizione. In generale quello che si osserva è una maggiore "smoothness" nei movimenti, dando all'agente un comportamento più lineare e meno brusco. Si noti ad esempio, in figura 43 la differenza di comportamento fra un agente con LSTM al suo interno, il primo, ed uno senza, addestrati per il resto nelle medesime condizioni. Dalle immagini in figura 43 non si nota benissimo perché



(a) *Test 1 della categoria 1 - Posizioni occupate dall'agente LSTM in tutto l'episodio.*

(b) *Test 1 della categoria 1 - Posizioni occupate dall'agente senza LSTM in tutto l'episodio.*

Figura 43: Grafico delle posizioni dei due agenti nel test 1 della categoria 1. In entrambi i casi il test è stato superato.

l'immagine è divisa in celle, ma nel caso a l'agente ha effettuato delle svolte curvando ed in generale ha coperto più spazio nella sua esplora-

zione.
Inoltre l'agente *b*, durante il suo viaggio verso l'angolo destro in alto ha puntato dritto verso l'angolo, senza perciò vedere cosa c'era nel resto dell'arena.
In definitiva i risultati migliori sono stati ottenuti proprio utilizzando insieme questi ultimi 5 approcci presentati, nella tabella 6 vengono riportati i tre punteggi più alti ottenuti nei test della competizione utilizzando gli approcci presentati, essi provengono dalla stessa configurazione di addestramento ma numero di step di update diversi.

Updates	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	Totale
91136000	23	17	4	10	6	15	15	3	9	0	34
101376000	23	16	1	11	5	15	16	3	10	0	33,33
52224000	23	15	2	12	4	14	14	4	11	0	33

Tabella 6: Risultati ottenuti nei test ufficiali utilizzano reward shaping, curriculum learning, divisione in quadranti dei colori rosso, giallo e verde, considerando la velocità dell'agente e utilizzando una LSTM.

Sicuramente addestrare una rete con al suo interno una LSTM richiede un numero maggiore di step di addestramento, ma questa maggiore complessità viene compensata dalla semplicità complessiva dell'approccio adottato.

5.3.8 *Utilizzare più modelli*

I risultati ottenuti con gli approcci presentati sono senz'altro ottimi se considerata la complessità dei test, ma difficilmente gli scenari più avanzati vengono superati con successo, sopratutto a causa delle semplificazioni che si sono fatte e di cui si è discusso in precedenza.
In base alle scelte intraprese per addestrare gli agenti si sono ottenuti risultati diversi:

- Se si è scelto di considerare molti colori nel calcolo dell'input si avranno migliori risultati nella categoria degli ostacoli, ma peggiori in quelle più semplici.
- L'utilizzo di reti ricorrenti può favorire l'agente in alcuni test ma sfavorirlo in altri.
- Ovviamente anche l'architettura di rete influisce sui risultati.
- La funzione utilizzata per il reward shaping può favorire la prestazione in alcune categorie e peggiorarla in altre.

Quindi per massimizzare i risultati si potrebbe pensare di valutare quali siano i modelli migliori per ogni categoria, ovvero quali modelli riescano ad esprimere una particolare capacità cognitiva, ed utilizzare

uno piuttosto dell'altro a seconda della situazione. L'intuizione è sicuramente sensata, in quanto riflette anche la divisione in aree funzionali che sembra esserci nel cervello delle specie evolute.
La difficoltà in questo caso risiede però nel come decidere quale modello mettere in campo in base alla situazione, sviluppare una strategia efficace non sarebbe stato compatibile con le tempistiche della competizione ma potrebbe rappresentare un elemento da investigare in futuri sviluppi. Data questa difficoltà si potrebbero mettere in campo due strategie:

1. Prendere per ogni categoria il modello che ha ottenuto il miglior risultato, disponendo così di 10 modelli. Quando l'agente deve scegliere l'azione da fare viene preparato un input apposito per ognuno dei modelli, i quali proporranno ognuno una azione da compiere. A questo punto si applica un meccanismo vote-based, in cui l'azione più votata verrà selezionata. Per il meccanismo di voto si potrebbe pesare la preferenza rispetto al punteggio complessivo ottenuto nei test della competizione.

2. Prendere per ogni categoria il modello che ha ottenuto il miglior risultato, disponendo così di 10 modelli come nel caso precedente. Se però l'agente conosce a quale categoria appartiene il test può automaticamente utilizzare il modello migliore per essa. Nel caso della fase principale della competizione, i test vengono eseguiti in maniera sequenziale, dalla categoria 1 alla 10. Con 30 test per ogni categoria. E' perciò facile ottenere questa informazione.

Come facilmente intuibile dalla tabella 7, i risultati migliori vengono ottenuti dall'approccio 2, in quanto di fatto si ottiene la somma dei migliori punteggi per ogni categoria ottenuti dai diversi modelli.

Agente	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	Totale
Agente-2	25	20	7	12	10	17	15	7	10	2	41.67
Agente-1	25	14	1	10	3	14	8	1	8	0	28

Tabella 7: Risultati ottenuti nei test ufficiali utilizzando più modelli. L'agente 1 utilizza la strategia 1, e l'agente 2 la 2.

A prescindere da questo risultato, che sfrutta evidentemente la modalità di valutazione scelta dagli organizzatori della competizione, è interessante vedere quali siano i modelli che hanno ottenuto i migliori risultati nelle 10 categorie:

- Per le categorie 1 e 2 degli agenti che considerano solo i colori verde, giallo e rosso, dividendo l'immagine in 49 quadranti e che non utilizzano LSTM, ottengono i risultati migliori. Questo perché probabilmente in queste due semplici categorie sono preferibili approcci minimali, che mappino i semplice stimoli di attrazione verso determinati oggetti.

- Per le categorie 3 e 4 lo stesso modello che considera i colori verde, giallo, rosso, grigio, rosa e blu, dividendo l'immagine in 49 quadranti e che non utilizzano LSTM, ottiene i risultati migliori. Questo perché nella categoria 3 è richiesto un riconoscimento almeno elementare dei vari oggetti presenti per potere superare i test, cosa che può essere possibile solo considerando i rispettivi colori. Il successo dello stesso modello nella quarta categoria è legata probabilmente al fatto che l'agente ha imparato in generale ad evitare ostacoli e le zone mortali possono essere viste come tali.

- Per le categorie 5, 6, 7 e 9 è stato un modello che considera solo i colori verde, giallo e rosso, dividendo l'immagine in 49 quadranti e che utilizza una LSTM, ad ottenere i risultati migliori. In effetti in questi task è richiesta una capacità di mantenere delle informazioni dal passato.

- I risultati nelle categorie 8 e 10 sono stati ottenuti i risultati migliori da degli agenti che considerano solo i colori verde, giallo e rosso, dividendo l'immagine in 49 quadranti e che non utilizzano LSTM. In questo caso la migliore performance potrebbe anche non essere dovuta a proprietà particolari dell'agente ma frutto di particolari situazioni.

Sebbene il risultato ottenuto con l'approccio multi-modello sia davvero notevole, si deve tenere a mente che un cambio dei test o del loro numero produrrebbe risultati sicuramente peggiori e perciò questo tentativo deve esser visto più come una dimostrazione di cosa si potrebbe fare riuscendo a utilizzare il miglior modello in base alla situazione, piuttosto che una soluzione definitiva ed applicabile.

6

RISULTATI

Dopo aver esposto ed analizzato i vari approcci messi in campo (e relative basi teoriche) per affrontare le sfide poste dalla competizione Animal-AI Olympics, in questo capitolo saranno presentati i risultati ottenuti nelle varie fasi della competizione, con particolare attenzione al risultato finale.
Come spiegato nella sezione 3.3.5 la competizione è stata divisa in due macro-fasi: la prima parte dal 8 luglio al 1 settembre 2019 e la seconda parte dal 2 settembre al 1 novembre 2019.

6.0.1 *Risultato nella prima fase*

Per la prima parte la classifica è stata stilata sulla base dei risultati ottenuti attraverso i test automatici presenti sulla piattaforma EvalAI, con 300 test divisi nelle 10 categorie.
In questa prima parte della competizione ci si è classificati 9° su circa 60 partecipanti, con il seguente punteggio:

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	Totale
23	16	0	9	4	14	9	1	9	0	28.33

Tabella 8: Risultato migliore ottenuto fino al 1 settembre, valido per il 9°posto in classifica

Questo risultato è stato ottenuto da un agente addestrato dividendo l'immagine in 36 quadranti, considerando solo i colori rosso, giallo, verde, utilizzando una LSTM ma non la tecnica del reward shaping.
Il punteggio ottenuto, e la conseguente nona posizione, ha valso la conquista del premio di 500 dollari in crediti Amazon AWS.

6.0.2 *Risultato fra la prima e la seconda fase*

Nel periodo compreso fra la il termine della prima fase della competizione e la sua chiusura, gli organizzatori, per permettere di valutare la bontà del proprio approccio hanno messo a disposizione gli stessi soliti test sulla piattaforma EvalAI. Si ricordi infatti che i test sono nascosti

e l'unico modo di capire quanto bene si stia facendo è sottomettere l'agente alla piattaforma e vedere i risultati.
Per i risultati migliori in questa fase bisogna fare un distinguo fra quello ottenuto con l'approccio multi-modello illustrato nella sezione 5.3.8 e quello ottenuto con un singolo modello:

Agente	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	Totale
Multi-modello	25	20	7	12	10	17	15	7	10	2	41.67
Modello singolo	23	17	4	10	6	15	15	3	9	0	34

Tabella 9: Risultati ottenuti nei test ufficiali utilizzando un approccio multi modello e con modello singolo

Il punteggio di 41.67%, come visibile nella figura 44 è valso il quarto posto nella classifica presente su EvalAI[1], mentre il punteggio di 34% con i punteggi attuali otterrebbe un decimo posto. I partecipanti a questa fase della competizione erano 64.

Rank	Participant team	Total	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	Last submission at
1	ironbar	42.67	25.00	15.00	13.00	14.00	11.00	16.00	16.00	7.00	11.00	0.00	26 days ago
2	Trrrrr	42.67	25.00	18.00	10.00	12.00	9.00	16.00	15.00	6.00	12.00	5.00	28 days ago
3	sirius	42.00	25.00	16.00	8.00	9.00	12.00	18.00	18.00	4.00	16.00	0.00	28 days ago
4	UniboTeam	41.67	25.00	20.00	7.00	12.00	10.00	17.00	15.00	7.00	10.00	2.00	26 days ago
5	Oltau.ai	37.00	25.00	17.00	7.00	6.00	6.00	16.00	16.00	3.00	15.00	0.00	1 month ago
6	BronzeBlood	36.00	24.00	21.00	5.00	7.00	10.00	15.00	14.00	0.00	12.00	0.00	2 months ago
7	sungbinchoi	35.00	23.00	14.00	7.00	10.00	5.00	17.00	13.00	3.00	11.00	2.00	26 days ago

Figura 44: Cattura di schermata relativa alla classifica della AnimalAI-Competition nei test canonici utilizzati in tutta la competizioni.

6.0.3 *Risultato finale*

La classifica finale della competizione è stata stilata eseguendo dei test leggermente diversi da quelli utilizzati per il resto della competizione. Essi sono delle leggere variazioni, sia in numero che nelle configurazioni, di quelli originali. Anche in questo caso i test sono nascosti e per di più non è noto nemmeno il numero di test per categoria.
Nella tabella 10 sono riportate le prime dieci posizioni della classifica finale[2]. Il nostro team si è classificato nono, con un punteggio di 32.6%, il modello proposto per la valutazione in questo caso è stato quello multi-modello già discusso in precedenza. I risultati ottenuti sono ovviamente più bassi rispetto a quelli ottenuti nei test usati nel resto della competizione, proprio a causa della natura di questo approccio,

1 Disponibile all'url https://evalai.cloudcv.org/web/challenges/challenge-page/396/leaderboard/1107
2 La classifica completa è visualizzabile all'indirizzo http://www.animalaiolympics.com/results.html

che si basa sull'utilizzo dei risultati migliori per ogni categoria. E' però ragionevole pensare che l'approccio mono-modello avrebbe più o meno conseguito lo stesso punteggio.

Team	Totale	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Trrrrr	43.7	87.8	72.2	30.0	44.4	42.2	54.4	57.8	10.0	32.2	5.6
ironbar	43.6	87.8	63.3	40.0	48.9	40.0	34.4	51.1	25.6	43.3	1.1
sirius	38.7	92.2	66.7	23.3	26.7	34.4	36.7	50.0	6.7	48.9	1.1
BronzeBlood	35.4	73.3	81.1	17.8	21.1	30.0	40.0	38.9	2.2	50.0	0.0
Oltau.ai	35.0	71.1	73.3	21.1	24.4	24.4	33.3	46.7	5.6	50.0	0.0
sungbinchoi	34.3	84.4	56.7	23.3	27.8	31.1	42.2	28.9	1.1	36.7	11.1
DeepFox	33.2	82.2	45.6	24.4	34.4	24.4	27.8	45.6	12.2	34.4	1.1
ARF-RL	32.7	81.1	52.2	14.4	31.1	25.6	27.8	44.4	10.0	40.0	0.0
UniboTeam	**32.6**	73.3	75.6	13.3	21.1	23.3	32.2	36.7	8.9	40.0	1.1
Juramaia	32.0	84.4	71.1	12.2	31.1	13.3	33.3	30.0	10.0	32.2	2.2

Tabella 10: Prime 10 posizioni della classifica finale della competizione, sono state omesse per brevità le 54 posizioni successive. In questo caso tutti i risultati sono riportati in percentuale di task superati.

Al termine della competizione oltre ai primi 3 classificati, sono stati attribuiti dei premi di 200 dollari ciascuno a chi avesse ottenuto il miglior punteggio in una delle categorie della competizione ma che non avesse già vinto un altro dei premi.
Il nostro team, nella categoria 2 ha ottenuto il secondo miglior punteggio, ovvero 75.6%, dietro al miglior punteggio di 81.1% segnato dal team *BronzeBlood* (che però aveva già vinto il premio per un altra categoria). Il premio in denaro di 200 dollari è quindi andato proprio al nostro team.
In definitiva anche i risultati ottenuti in questa fase sono senz'altro buoni, anche considerando che in generale i punteggi finali non sono stati molto alti, a prova del fatto che la competizione poneva delle sfide davvero difficili da superare.

7

CONCLUSIONI E SVILUPPI FUTURI

Obiettivo di questa tesi era affrontare le complesse sfide poste dalla competizione Animal-AI Olympics, utilizzando un algoritmo allo stato dell'arte del reinforcement learning quale PPO, valutando diverse tecniche comuni nella materia, al fine di ottenere un agente capace di dimostrare il maggior numero possibile di capacità cognitive.
Dopo aver introdotto le basi teoriche del RL e di PPO nei capitoli 1 e 2, ed analizzato a fondo tutti gli aspetti della competizione nel capitolo 3, sono state condotte delle indagini esplorative sulle **tecniche di SLAM** a partire da immagini monoculari RGB, al fine di transitare verso un approccio di **RL Model-Based**. Date le tempistiche della competizione si è deciso di non utilizzare le soluzioni proposte in questo capitolo in quanto non del tutto mature, seppur potrebbero rappresentare degli sviluppi futuri davvero interessanti.
Nel capitolo 5 si è mostrato come applicare **PPO** all'ambiente della competizione, evidenziando come però un utilizzo semplicistico dell'algoritmo su immagini RGB pure, senza nessun altro tipo di accorgimento, non riesca a produrre risultati soddisfacenti e quindi come sia necessario mettere in campo delle strategie oculate per migliorare i risultati. Nello specifico si è proposto di semplificare lo spazio delle osservazioni dell'agente, considerando solo gli elementi di maggior interesse, ovvero quelli che attribuiscono premi e penalità. Si è quindi dimostrato che l'algoritmo durante l'addestramento impara ad associare le giuste sequenze di azioni in base agli stimoli che riceve, venendo attratto dagli elementi che attribuiscono premi e respinto da quelli che infliggono penalità.
Si è inoltre mostrata l'applicazione del concetto di **curriculum learning**, facendo addestrare l'agente prima su situazioni più semplici per poi fargli saggiare anche ambienti più probanti.
E' stato inoltre dimostrato che dividendo in quadranti l'immagine ottenuta da un agente ad ogni istante temporale si possono ottenere dati di input meno rumorosi e più significativi, permettendo un aumento delle performance.
L'utilizzo delle informazioni sulla velocità percepita dall'agente hanno permesso l'applicazione di tecniche di **reward shaping** per spingere l'agente verso comportamenti più desiderabili, producendo in effetti buoni risultati e comportamenti interessanti. Infine l'utilizzo di reti ricorrenti e di approcci multi-modello hanno permesso di ottenere modelli

ancora più prestanti nei vari test della competizione.
Al fine di addestrare e testare tutti i vari approcci presentati si è sviluppata una solida ed estendibile architettura sia per l'addestramento di agenti di RL sia per il loro testing locale. A questo scopo sono state inoltre progettate più di 150 configurazioni di arene (dell'ambiente della competizione), secondo i principi enunciati nelle sezioni 5.2.5 e 5.2.2.

In definitiva l'ottimo 9°posto su 64 partecipanti ottenuto nei test finali, in combinazione con il premio vinto per il miglior punteggio ottenuto sulla categoria atta a testare la capacità di esibire delle **preferenze**, dimostrano la bontà complessiva degli approcci utilizzati.
Ovviamente la capacità di generalizzazione dell'agente finale è ancora limitata, in quanto i risultati migliori sono stati ottenuti in quelle categorie dove le capacità cognitive da dimostrare erano meno complesse da ottenere. Guardando ai risultati complessivi della competizione, a prescindere da quelli ottenuti attraverso questo lavoro, si può sicuramente affermare che tutti approcci messi in campo dai partecipanti alla competizione di tutto il mondo non hanno permesso di avvicinarsi davvero al grado di performance ottenuta dagli animali più intelligenti. Sarà perciò necessario molto lavoro nel campo dell'AI in generale, e non solo del RL, per riuscire a fornire ad un singolo agente tutte le capacità cognitive richieste per superare task del genere.
Come sviluppi futuri del lavoro svolto sarebbe molto interessante innanzitutto trovare un metodo efficace per mappare ambienti a partire da immagini monoculari, fermo restando l'esigenza di disporre di metodi il più semplici e veloci possibili, in modo da non appesantire troppo il processo decisionale dell'agente. Disporre di una modellazione dell'ambiente accurata potrebbe permettere di transitare verso il RL-Model Based, il che potrebbe giovare per ottenere capacità cognitive quali ragionamento spaziale, percezione della permanenza degli oggetti e ragionamento causale.
Altro sviluppo verso cui si potrebbe andare è quello di identificare degli specifici modelli per ogni capacità cognitiva, mettendoli in campo a seconda della situazione in maniera totalmente automatica. La divisione in aree funzionali è un concetto presente nelle scienze cognitive e che può essere senz'altro sfruttato, fermo restando la difficoltà nello scegliere quali modelli utilizzare a partire dall'ambiente in cui è immerso l'agente.

BIBLIOGRAFIA

[1] Mordechai Ben-Ari and Francesco Mondada. *Robotic Motion and Odometry*, pages 63–93. Springer International Publishing, Cham, 2018. ISBN 978-3-319-62533-1. doi: 10.1007/978-3-319-62533-1_5. URL https://doi.org/10.1007/978-3-319-62533-1_5.

[2] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM, 2009.

[3] Benjamin Beyret, Jos'e Hern'andez-Orallo, Lucy Cheke, Marta Halina, Murray Shanahan, and Matthew Crosby. The animal-ai environment: Training and testing animal-like artificial cognition. 2019.

[4] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[5] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[6] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. https://github.com/openai/baselines, 2017.

[7] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE international conference on computer vision*, pages 2650–2658, 2015.

[8] Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I Jordan, Joseph E Gonzalez, and Sergey Levine. Model-based value estimation for efficient model-free reinforcement learning. *arXiv preprint arXiv:1803.00101*, 2018.

[9] Lex Fridman. Lecture notes of introduction to deep reinforcement learning, January 2019.

[10] Clément Godard, Oisin Mac Aodha, and Gabriel J Brostow. Unsupervised monocular depth estimation with left-right consistency. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 270–279, 2017.

[11] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

[12] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. https://github.com/hill-a/stable-baselines, 2018.

[13] Heiko Hirschmuller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on pattern analysis and machine intelligence*, 30(2):328–341, 2007.

[14] Baichuan Huang, Jun Zhao, and Jingbin Liu. A survey of simultaneous localization and mapping, 2019.

[15] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[16] Arthur Juliani, Vincent-Pierre Berges, Esh Vckay, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*, 2018.

[17] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *ICML*, volume 2, pages 267–274, 2002.

[18] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[19] Andrei Andreevich Markov. The theory of algorithms. *Trudy Matematicheskogo Instituta Imeni VA Steklova*, 42:3–375, 1954.

[20] James L McClelland, Bruce L McNaughton, and Randall C O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review*, 102 (3):419, 1995.

[21] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2, 2014.

[22] Tom Michael Mitchell. *The discipline of machine learning*, volume 9. Carnegie Mellon University, School of Computer Science, Machine Learning ..., 2006.

[23] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[24] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[25] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[26] Arsalan Mousavian, Hamed Pirsiavash, and Jana Košecká. Joint semantic segmentation and depth estimation with deep convolutional networks. In *2016 Fourth International Conference on 3D Vision (3DV)*, pages 611–619. IEEE, 2016.

[27] Matthias Ochs, Adrian Kretz, and Rudolf Mester. Sdnet: Semantically guided depth estimation network. In *German Conference on Pattern Recognition*, pages 288–302. Springer, 2019.

[28] OpenAI. Proximal policy optimization. https://openai.com/blog/openai-baselines-ppo/, 2017.

[29] OpenAI. Openai five. https://blog.openai.com/openai-five/, 2018.

[30] OpenAI. A taxonomy of rl algorithms, 2019. URL https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html#a-taxonomy-of-rl-algorithms.

[31] Joseph O'Neill, Barty Pleydell-Bouverie, David Dupret, and Jozsef Csicsvari. Play it again: reactivation of waking experience and memory. *Trends in neurosciences*, 33(5):220–229, 2010.

[32] Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, Vladlen Koltun, and Dawn Song. Assessing generalization in deep reinforcement learning. *arXiv preprint arXiv:1810.12282*, 2018.

[33] Federico Pedersini. Slide di visione stereoscopica, corso di visione artificiale, January 2019.

[34] O. Ronneberger, P.Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 9351 of *LNCS*, pages 234–241. Springer, 2015. URL http://lmb.informatik.uni-freiburg.de/

Publications/2015/RFB15a. (available on arXiv:1505.04597 [cs.CV]).

[35] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[36] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[37] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[38] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[39] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[40] Herbert A Simon. The sciences of the artificial mit press. *Cambridge, MA*, 1969.

[41] Herbert Alexander Simon. *The shape of automation for men and management*, volume 13. Harper & Row New York, 1965.

[42] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[43] Csaba Szepesvári. Algorithms for reinforcement learning. *Synthesis lectures on artificial intelligence and machine learning*, 4(1):1–103, 2010.

[44] István Szita and András Lörincz. Learning tetris using the noisy cross-entropy method. *Neural computation*, 18(12):2936–2941, 2006.

[45] Tensorflow. Image segmentation. https://www.tensorflow.org/tutorials/images/segmentation, 2019.

[46] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[47] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, pages 1–5, 2019.

[48] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

[49] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016.

[50] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[51] Wikipedia. Computer stereo vision. https://en.wikipedia.org/wiki/Computer_stereo_vision, 2018.

[52] Deshraj Yadav, Rishabh Jain, Harsh Agrawal, Prithvijit Chattopadhyay, Taranjeet Singh, Akash Jain, Shiv Baran Singh, Stefan Lee, and Dhruv Batra. Evalai: Towards better evaluation systems for ai agents. *arXiv preprint arXiv:1902.03570*, 2019.

RINGRAZIAMENTI

Giunto alla fine di questo lavoro di testi e del mio percorso universitario vorrei fare diversi ringraziamenti.
In primo luogo ci tengo a ringraziare il prof. Davide Maltoni ed il mio correlatore Vincenzo Lomonaco, che mi hanno dato la possibilità di svolgere una tesi su un argomento così interessante e stimolante, fornendomi sempre ottimi spunti di riflessione e possibili soluzioni, guidandomi nel percorso di tesi.
Ringrazio molto anche i ragazzi del Biolab di Cesena, in particolare Gabriele Graffieti che mi ha messo a disposizione il server per fare esperimenti e che mi ha sapientemente consigliato riguardo alle tematiche del capitolo 4.
Prima di passare agli affetti più cari vorrei fare un ringraziamento generale ai professori ed allo staff in generale del mio corso di laurea, grazie a voi sono davvero soddisfatto di questo percorso durato 5 anni.
Ed ora un ringraziamento davvero grande a tutte persone a me più care:

Ai miei genitori, che mi hanno sempre supportato in qualsiasi modo possibile, mettendomi sempre al centro della loro vita. Grazie per tutto quello che avete fatto per me, siete due persone davvero speciali, sono fiero di avere due genitori così.
Alla mia ragazza Ilaria, che in questi ultimi due anni e mezzo mi è sempre stata accanto, dandomi conforto quando ne avevo bisogno, facendomi passare momenti bellissimi insieme e rispettando sempre i miei studi e le mie passioni. Grazie, penso di essere davvero fortunato ad averti accanto.
A zio Davide, che mi è sempre stato vicino, mi ha supportato in qualsiasi modo, mi ha consigliato, mi ha fornito spunti di riflessioni e mi spinge sempre ad essere una persona migliore.
Alla mia famiglia tutta, i miei nonni, tutti i miei zii e zie.
Ai miei amici storici Eric, Casto e Dero, nonostante siamo 4 persone davvero diverse l'una dall'altra con voi sono cresciuto, ho sempre passato dei momenti stupendi e penso di aver imparato davvero tanto insieme a voi. Grazie per esserci sempre.
A Luca, il mio coinquilino, amico, compagno di università e di calcetti di questi 5 anni, è anche grazie a te che ricorderò questi anni di università come anni fantastici. Sei una persona profondamente buona e ti ringrazio per tutto.
A Franco, Laura, Lucia e Chicco, la mia seconda famiglia in questi ultimi 4 anni, non avrò mai parole per ringraziarvi abbastanza, siete delle persone stupende ed un'ispirazione per me, se sono arrivato alla fine di questo percorso con questi risultati lo devo anche a voi.

Agli amici conosciuti all'università (ed ai calcetti): Joseph, Diego, Bocc, Marco, Magno, Davide, Marco, Turi, Placu, Andrea per citarne alcuni, vi ringrazio per aver reso questo percorso molto più divertente e stimolante, siete tutte persone davvero in gamba e farete sicuramente grandi cose.
La lista sarebbe ancora più lunga, perché sono davvero tante le persone che in questi anni mi hanno aiutato, supportato e fatto crescere, vorrei quindi ringraziarvi tutti dal profondo del cuore, anche se non comparite per nome non siete meno importanti.